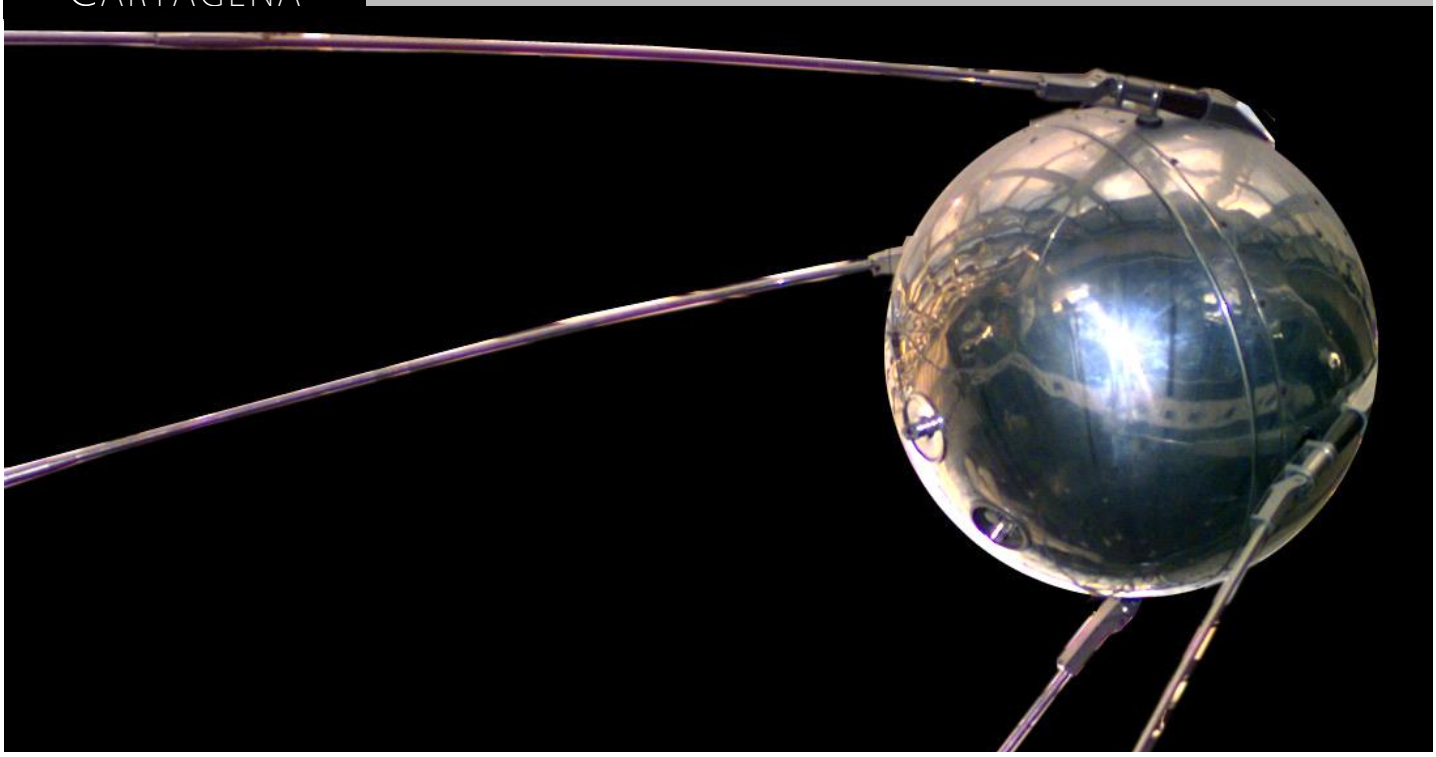
Universidad Politécnica de Cartagena

Creación de un sistema de control de rotores mediante Python



Alumno: Samuel Góngora García
Director: José Luis Gómez Tornero

Escuela Técnica Superior de Ingeniería Industrial

A mi familia, que me dio la oportunidad de estudiar esta carrera y a mi pareja, Alejandra, a la que debo tanto por su incondicional apoyo.

.

Agradecimientos.

Resulta difícil resumir un camino tan largo en una simple página. Para el que escribe estas líneas este proyecto es mucho más que un tocho generoso de páginas y varios años de trabajo. Este proyecto es un punto y aparte. Este proyecto es el comienzo de una nueva etapa en la vida.

Podríamos decir que esta historia empieza a la edad de cuatro años. Por aquel entonces empezó en mí una afición que más tarde se convertiría en una pasión que ha guiado gran parte de mi vida. A tan temprana edad decidí que la astronomía, y todo lo relacionado con ella, sería aquello a lo que dedicaría mi vida.

Así pasó la adolescencia hasta que llegaron los dieciocho y el momento de elegir una carrera. Con una gran dosis de idealismo y una pequeña dosis de frialdad decidí estudiar una ingeniería.
La ingeniería electrónica se postulaba como la mejor alternativa a un futuro lleno de naves, sondas y demás cachivaches espaciales.

Mis padres, Jose y Mª Carmen, me dieron la oportunidad, con su esfuerzo, de empezar mis estudios en otra ciudad. Mis dos hermanas mayores, Carolina y Cristina, supieron guiarme en aquella época, cuando andaba algo solo y perdido en mi nueva vida.
Hice buenos amigos en la Universidad Politécnica de Cartagena. Gente que amaba la ingeniería y que te motivaba a investigar y aprender más. Siempre recordaré aquellas tardes en las escaleras de la Escuela.

Después de una estancia en Alemania tocó el momento de volver a Cartagena. En aquel momento surgió una de las figuras que más apoyo me ha dado durante el transcurso de la carrera.
Presenté a José Luis Gomez Tornero una idea algo descabellada como proyecto final de carrera. Él, lejos de asustarse, la vio hasta viable y, de esta manera, empezó a perfilarse el proyecto que ahora entrego.

El último año de carrera quizás haya sido el más duro de toda ella. La realización del proyecto cada vez consumía más tiempo y esfuerzo y los exámenes más difíciles aún estaban por aprobar.
En este momento es cuando tengo que agradecer, con todo mi corazón, a Alejandra, mi pareja, por haberme apoyado durante este último año. Su presencia en los momentos más duros, cuando el peso de los años me aplastaba, me animó a seguir un día tras otro trabajando en mis ideas.
Desde aquí me gustaría hacerle saber que parte de las palabras de esta memoria le pertenecen.

Pero los exámenes se aprobaron y el trabajo en el proyecto llegó a su fin. Y ahora han surgido nuevos planes y el futuro se antoja prometedor. Una nueva vida en una nueva

ciudad en la que seguiré aprendiendo y evolucionando junto a las personas que me quieren.

Ahora pueden leer, condensado en este tomo, todo el proceso de investigación que me llevó a fabricar el prototipo que presento como proyecto final de carrera.
Y desde aquí no puedo sino agradecer, de nuevo, a todas aquellas personas que he nombrado en estas líneas sin las que nada de esto habría sido posible.

Índice

Listado de Figuras

Listado de Tablas

1 – Introducción.

Motivaciones de este proyecto.

En este proyecto final de carrera abordaremos la creación de un sistema de control de rotores destinados al seguimiento de satélites, en un principio, de órbita baja.
La placa de desarrollo utilizada en este proyecto será una Raspberry Pi. Como podremos ir comprobando con la lectura de esta memoria la utilización de elementos de hardware y software libres será un pilar de este proyecto.

La continua aparición de hardware para prototipado, y la gran comunidad de desarrolladores de software libre actual, es el ambiente propicio para la creación de un sistema de estas características.
Los varios millones de placas de desarrollo Raspberry Pi vendidas, y las decenas de desarrollos inspiradas en ésta, no son sino una muestra de la situación actual.

Sirvan estos dos párrafos como una introducción de las intenciones de este desarrollo. Más adelante, en el tercer capítulo de esta memoria, abordaremos con más profundidad el contexto actual que ha motivado este proyecto final de carrera.
También incidiremos en la posible influencia de éste en el panorama tecnológico actual.

Contenido de esta memoria.

En esta memoria abordaremos el proceso de creación de este prototipo. Describiremos los antecedentes de este proyecto, las motivaciones de éste, el desarrollo de la interfaz del programa y sus rutinas de cálculo, el conexionado al sistema de motores y las pruebas finales de rendimiento del conjunto de antena y receptor mediante este sistema de guiado.
También incluiremos un capítulo para mostrar futuras mejoras de nuestro prototipo.

Todo el código creado estará disponible en un anexo para su evaluación. Este mismo código estará disponible en el perfil[1] de Github del creador de este proyecto.

Estructura de la memoria por capítulos.

La estructura de esta memoria tratará de asignar un capítulo a cada paso del desarrollo del proyecto. Esto clarificará la comprensión de éste y facilitará la búsqueda de un punto en concreto. La organización será la siguiente.

[1] https://github.com/sgongar

1 – Introducción

En el primer capítulo detallaremos la temática del proyecto. En este punto daremos una visión somera de las motivaciones de este desarrollo. También intentaremos detallar como abordaremos su creación.

2 - Antecedentes.

Para tener una visión más clara del panorama actual intentaremos ofrecer una descripción de los proyectos actuales relacionados con la temática de este en concreto.
Enunciaremos desde los desarrollos oficiales hasta los desarrollos libres realizados por aficionados.

3 - Objetivos de este proyecto.

Una vez definidos los antecedentes de este proyecto, definiremos los objetivos a cumplir con este prototipo. Este capítulo será, además de una recopilación de las funcionalidades que tendremos que ser capaces de ofrecer, una declaración de intenciones del creador de este desarrollo.

4 - Componentes.

El funcionamiento del hardware y software elegido anteriormente será detallado en este capítulo. Así facilitaremos la comprensión de los siguientes puntos de esta memoria.

5 - GUI.

La creación de la interfaz gráfica se detallará en este capítulo. Trataremos de explicar todo el proceso, desde la elección de una librería adecuada hasta el resultado obtenido por pantalla.
La problemática asociada a la creación de un código multiplaforma será abordada aquí. Si surgiera algún problema trataríamos de solventarlo de la mejor manera posible.
El código creado para la interfaz gráfica estará disponible en el primer punto del anexo de esta memoria.

6 - Rutinas de cálculo.

Necesitaremos crear un código que, utilizando las librerías elegidas anteriormente, nos proporcione los datos necesarios.
El proceso de creación de este código será detallado en este capítulo.
En caso de que necesitemos más objetos para el funcionamiento del script también los incluiremos en este punto.
Todo el código fuente generado será incluido en el segundo apartado del anexo.

7 - Interfaz de conexión.

Abordaremos los problemas que nos puedan surgir al conectar nuestra placa de desarrollo con el sistema de motores elegido.
En el caso de necesitar alguna interfaz de tipo hardware el proceso de diseño y creación de ésta será abordado en este capítulo.
Todos los archivos que generemos en su desarrollo serán incluidos en el CD que acompaña a esta memoria.

8 - Conclusiones.

En este capítulo trataremos de sacar conclusiones sobre nuestro prototipo comprobando su funcionamiento. Estas pruebas de campo se realizarán mediante simulaciones o, cuando sea posible, implementando nuestro desarrollo en sistemas reales ya instalados.

9 - Mejoras.

Con los datos y las simulaciones obtenidas en el capítulo anterior daremos un listado de posibles mejoras del prototipo. Éstas podrán atañer tanto al software de éste, como al hardware utilizado.

Anexo.

En el anexo incluiremos el código fuente desarrollado en este proyecto. El anexo estará dividido en tantos puntos como ficheros tenga nuestro desarrollo.

Bibliografía.

Todas las referencias bibliográficas mencionadas en esta memoria estarán recogidas en este punto. Se incluyen las referencias a documentos de todo tipo, desde páginas web hasta libros de bibliografía básica.

Declaración de intenciones.

Espero que los que lean esta memoria disfruten al menos una décima parte de lo que yo he disfrutado con el proceso de creación de ésta.

Espero que los que aún no estén inmersos en el campo de la astronáutica, al leer este proyecto, consideren dar sus primeros pasos en ella.

Espero que los que ya estén inmersos en el tema vean mis primeros pasos con cariño y sepan tratar con benevolencia mis errores.

Dicho esto, comenzamos.

2 - Antecedentes.

Introducción.

En este capítulo procuraremos dar una visión general del panorama actual de la temática relacionada con nuestro proyecto.

Breve historia de los sistemas profesionales de seguimiento de satélites.

El panorama ha cambiado mucho desde que el Sputnik, allá por el año 1957, realizara su primera órbita gracias al esfuerzo de la Unión Soviética.

La cantidad de objetos en órbita ha crecido exponencialmente desde aquel entonces. A los satélites de las grandes agencias espaciales se les unieron, a partir del 1962, los primeros satélites para usos comerciales.

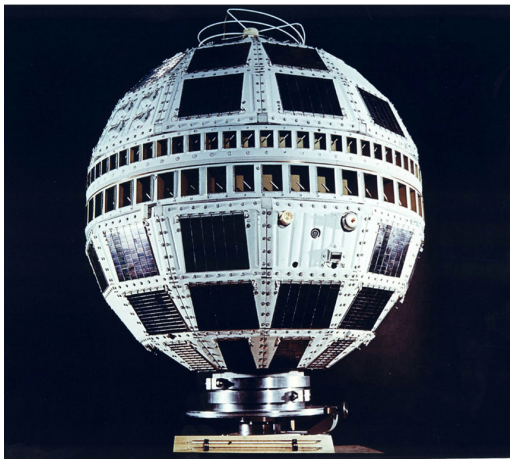


Figura 1. Satélite de telecomunicaciones Telstar (Smith, 2012)[2].

A todos los objetos que actualmente orbitan a la tierra cumpliendo una función, tendremos que sumarle todos aquellos satélites que dejaron de funcionar y, no menos importantes, las etapas impulsoras de estos que no retornaron a la Tierra y quedaron en órbita.

Esta gran cantidad de objetos es vigilada constantemente por varios organismos internacionales. El más importante de ellos es la red de vigilancia espacial del ejército estadounidense (siglas SSN).

[2] http://www.nasa.gov/topics/technology/features/telstar.html

La SSN lleva siguiendo desde 1957, mediante su red de telescopios y radares, todos los objetos espaciales mayores de diez centímetros, realizando entre 380.000 y 420.000 observaciones cada día.

Para hacernos una idea del trabajo que esto supone, mencionaremos que los últimos estudios estiman que existen más de 21.000 objetos en órbita de un tamaño superior a los diez centímetros.
Los objetos inferiores a este tamaño son actualmente irrastreables y se estima que rondan el medio millón (Autonomous Nonprofit Organization "TV-Novosti", 2013)[3].

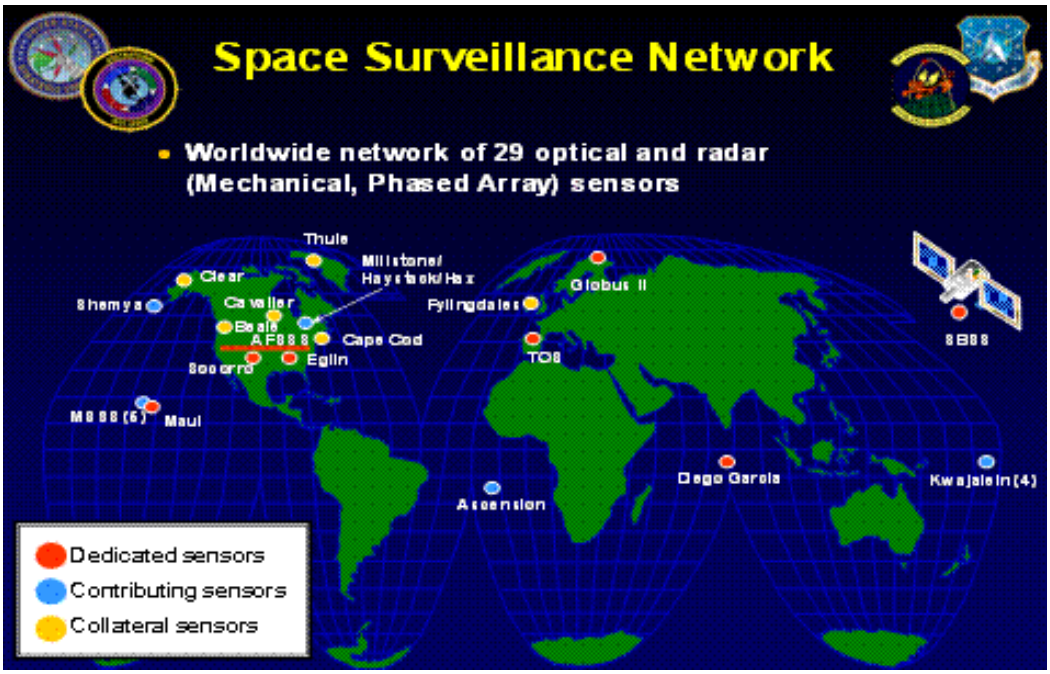


Figura 2. Red mundial de seguimiento de la SSN (SSN, 2012) [4]

La enorme cantidad de datos recolectados por la SSN (y otros organismos colaboradores) son puestos a disposición del público a través de internet.
La NASA, en conjunción con el NORAD, ofrece en estos servicios web los datos orbitales de los satélites rastreados mediante un formato, considerado un estándar, denominado TLE.

TLE es el acrónimo de *two-line element.* Un TLE consiste en un grupo de dos líneas de texto de 69 caracteres que contiene la información necesaria para, mediante el modelo orbital *SGP4/SDP4* (Felix R. Hoots, 1988)[5], obtener la posición y velocidad de un satélite para un instante dado.

[3] http://rt.com/usa/space-fence-shut-down-455/

[4] http://www.stratcom.mil/factsheets/USSTRATCOM_Space_Control_and_Space_Surveillance/

[5] http://celestrak.com/NORAD/documentation/spacetrk.pdf

La descripción de los diferentes elementos de un TLE es fácilmente accesible en la red. Se puede encontrar una descripción muy detallada de éstos realizada por el doctor T.S. Kelse en su sitio web (Kelso, CelesTrak: "FAQs: Two-Line Element Set Format", 2006)[6].

Estos grupos de dos líneas de caracteres se agrupan en archivos de texto mayores atendiendo a la naturaleza del satélite. Los satélites con funciones similares se agrupan en familias.
Algunas de estas familias son conocidas por el gran público como son, por ejemplo, los satélites de telecomunicaciones de la familia Iridium.

La evolución de la tecnología, y el abaratamiento de costes que esto conlleva, han creado numerosos aficionados que se dedican, de manera no profesional, al seguimiento de todo tipo de satélites artificiales. Esto ha provocado el aumento de la popularidad de los sitios dedicados a suministrar datos orbitales actualizados.

Para la realización de este proyecto hemos estudiado la posibilidad de usar alguno de los dos sitios siguientes. Aunque ambos ofrecen directorios actualizados de elementos TLE cuentan con diferencias significativas entre ellos que merece la pena sopesar.

Celestrak.

Celestrak[7] es un sitio web mantenido por el doctor Thomas Sean Kelso (en adelante T.S. Kelso) (Kelso, Dr. T.S. Kelso, CelesTrak WWW, 2013)[8].

Celestrak ofrece en su sitio web, además de elementos TLE, acceso a otros datos orbitales de interés y diversa documentación sobre esta temática. Actualmente es el sitio web más veterano dedicado a esta tarea.

Los elementos TLE de cada familia están recogidos en un fichero, con formato txt, con el nombre de la familia en cuestión.
El acceso a estos ficheros es libre y no requiere de registro alguno (Kelso, CelesTrak: Current NORAD Two-Line Element Sets, 2013)[9]. Los ficheros disponibles en este servicio se agrupan dependiendo de las diferentes naturalezas de los objetos. Atendiendo a ese criterio la distribución de los elementos orbitales es la siguiente:

- Special-Interest Satellites. *Satélites con un especial interés.*
    - Last 30 Days' Launches. *Lanzamientos de los útlimos 30 días.*
    - Space Stations. *Estaciones espaciales.*
    - 100 (or so) Brightest. *Los 100 más brillantes.*

[6] http://celestrak.com/columns/v04n03/
[7] http://celestrak.com/
[8] http://celestrak.com/webmaster.asp
[9] http://celestrak.com/NORAD/elements/

- FENGYIUN 1C Debris
- IRIDIUM 33 Debris
- COSMOS 2251 Debris
- BREEZE-M R/B Breakup (2012-044C)

- Weather & Earth Resources Satellites. *Satélites de recursos meteorológicos y terrestres.*

  - Weather. *Clima.*
  - NOAA.
  - GOES.
  - Earth Resources. *Recursos terrestres.*
  - Search & Rescue. (SARSAT). *Busqueda y rescate.*
  - Disaster Monitoring. *Control de desastres.*
  - Tracking and Data Relay Satellite System. (TDRSS). *Sistema de satélites para la comunicación y seguimiento espacial.*

- Communications Satellites. *Satélites de comunicaciones.*

  - Geostationary. *Geoestacionario.*
  - Intelsat.
  - Gorizont.
  - Raduga.
  - Molniya.
  - Iridium.
  - Orbcomm.
  - Globalstar.
  - Amateur Radio. *Radio amateur.*
  - Experimental. *Experimental.*
  - Other. *Otros.*

- Navigation Satellites. *Satélites para navegación.*

  - GPS Operational. *Operativa GPS.*
  - Glonass Operational. *Operativa Glonass.*
  - Galileo.
  - Beidou.
  - Satellite-Based Augmentation System. (WAAS/EGNOS/MSAS). *Sistema de satélites para la mejora del posicionamiento.*
  - Navy Navigation Satellite System. (NNSS). *Sistema de satélites para la navegación de la armada.*
  - Russian LEO Navigation. *Navegación rusa mediante LEO.*

- Scientific Satellites. *Satélites científicos.*
  - Space & Earth Science. *Ciencia espacial y terrestre.*
  - Geodetic. *Geodésica.*
  - Engineering. *Ingeniería.*
  - Education. *Educación.*

- Miscellaneous Satellites. *Satélites diversos.*
  - Miscellaneous Military. *Satélites diversos militares.*
  - Radar Calibration. *Calibración de radar.*
  - CubeSats.
  - Other. *Otros.*

Figura 3. Página web principal del sitio http://celestrak.com/

La ventaja de un acceso libre a los datos se contrapone al hecho de la imposibilidad de obtener el TLE de un elemento en particular.
Podremos implementar un algoritmo que seleccione un objeto en concreto de la lista pero esto redundará en más trabajo por parte del script y una mayor ineficiencia de este.

Space-track.

Space-track es un sitio web, de más reciente creación, que también ofrece un servicio actualizado de elementos orbitales. La gran diferencia existente entre uno y otro es que Space-track requiere registro por parte del usuario.
Una vez realizado el registro, tras esperar la aceptación de este por parte de Space-Track, se podrá acceder a la ventana principal del servicio.

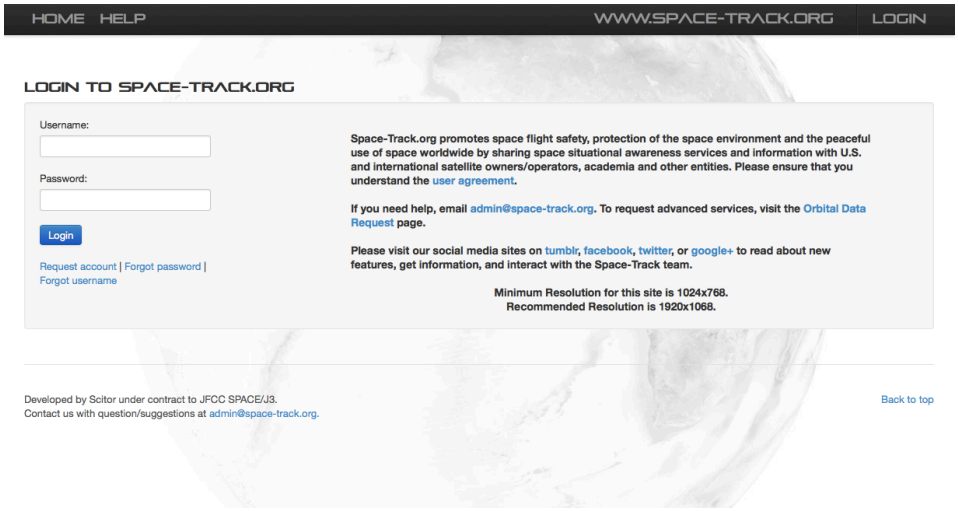


Figura 4. Pantalla de acceso a Space-Track[10].

Las opciones que nos brinda este sitio web son mucho más completas que las proporcionadas por Celestrak. A continuación las detallaremos una a una.

- Box Score.
  - En esta pestaña podremos acceder a un listado donde aparecerá el número de objetos (en órbita o no) de cada país.
- SATCAT.
  - En esta pestaña encontraremos un listado, en forma de tabla, de todos los objetos lanzados al espacio. Estarán mezclados aquellos que aún orbitan a la tierra con los que ya reentraron en la atmosfera.
- Decay/Reentry.
  - En la siguiente pestaña se encontrarán únicamente aquellos objetos que reentraron en la atmosfera o tienen previsto hacerlo.
- Query Builder.
  - Desde aquí podremos crear una instancia con la API proporcionada por este sitio web. Esta API nos permitirá crear una web personalizada donde aparecerán los elementos orbitales de los objetos deseados.
- Favorites.
  - Aquí tendremos la posibilidad de crear un listado de objetos favoritos. Se podrán crear listas con selecciones personalizadas, no es necesario que sean elementos de países o familias similares.
- TLE Search.
  - La búsqueda de elementos particulares se realizará desde esta ventana. Para identificar un objeto se tendrá que dar su nombre, designación internacional o número de identificador.
- Recent TLEs.
  - En esta ventana encontraremos una colección de elementos TLE organizados por familias. Aunque el sistema de organización es similar al utilizado por Celestrak, las familias serán algo diferentes. Las denominaciones utilizadas por Space-Track serán las siguientes:

[10] https://www.space-track.org

- Full Catalog. *Catálogo completo.*
- Geosynchronous. *Geosíncronos.*
- Navigation. *Navegación.*
- Weather. *Clima.*
- Iridium.
- Orbcomm.
- Globalstar.
- Intelsat.
- Inmarsat.
- Amateur.
- Visible.
- Special Interest. *Especial interés.*
- Bright Geosyncronous. *Geosíncronos brillantes.*
- Human Spaceflight. *Vuelos humanos espaciales.*

Otro cambio con respecto a Celestrak es la posibilidad de descargar los elementos orbitales en un formato de dos o tres líneas. En el formato de tres líneas, la primera línea corresponderá al nombre del satélite. Esto puede ser útil para identificar qué elementos corresponden a qué objeto.

- SSR.
  - Aunque esta pestaña no ofrece ningún elemento orbital posee algunas características que la hacen interesante. En ella podemos encontrar diferentes listados, en formato html y csv, de los objetos lanzados al espacio.

Futuro.

En un mundo en el cual los lanzamientos de satélites no dejarán sino de aumentar, se antojan cada vez más necesarios servicios como los anteriormente mostrados.
Este proyecto sólo pretende aportar su grano de arena al desarrollo tecnológico que nos espera en el campo espacial.

Desarrollos libres actuales.

Trataremos de dar una visión general del panorama 'open source' actual centrándonos en nuestro terreno en particular.

Software de guiado actual.

Una de las grandes taras existentes en el panorama de sistemas de guiado es la falta de desarrollos de software libre en este campo. Este panorama empeora si, además, buscamos un desarrollo que sea multiplataforma y fácilmente configurable por el usuario.
Podremos también dividir estos desarrollos en programas complemente desarrollados y librerías para su uso en programas o rutinas de cálculo.

Pasaremos, a continuación, a mencionar aquellos proyectos que consideremos más relevantes.

Programas.

Entendiendo programa como desarrollo completo utilizable tras su instalación.

JsatTrak.

JSatTrack es un programa para el seguimiento de satélites desarrollado en Java.

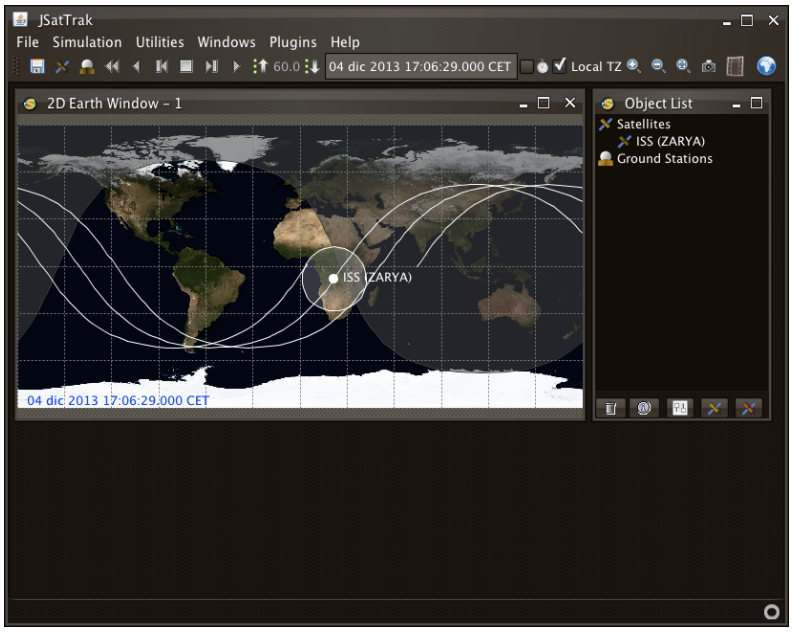


Figura 5. Ventana principal de JSatTrack.

Es un programa altamente personalizable que nos permitirá definir que queremos exactamente de él.

Algunas de las opciones que nos brinda, son:

- Obtención de efemérides en tiempo real o en un instante dado.
- Uso remoto de la aplicación vía internet.
- Permite la definición de más de ochocientas estaciones base.
- Carga de elementos TLE desde Celestrak o desde ficheros de texto.
- Animaciones en 2D y en 3D de la trayectoria de los satélites.

El código fuente de este desarrollo se encuentra disponible en un repositorio de GitHub[11] bajo una licencia Apache 2.0 (The Apache Software Foundation, 2012)[12].

[11] https://github.com/sgano/JSatTrak
[12] http://www.apache.org/licenses/LICENSE-2.0

Aunque JSatTrack es un desarrollo muy completo que nos proporciona una gran cantidad de información tiene una gran tara, y es que no controla sistemas de rotores.

PREDICT.

PREDICT ha sido, y es, la base de muchos desarrollos actuales en este campo.
Los datos que nos proporciona este programa son los enumerados a continuación:

- Acimut y elevación del satélite.
- Valor del efecto doppler.
- Dispersión de la señal del satélite.
- Distancia del satélite a la estación receptora.
- Altitud y velocidad orbital.
- Huella satelital.
- Anomalía media de la órbita.
- Ángulo entre las antenas del satélite y la estación receptora.
- Profundidad del eclipse generado por la tierra sobre el satélite.
- Número de órbita del objeto.

PREDICT también incorpora las rutinas necesarias para el guiado de rotores con una gran precisión.

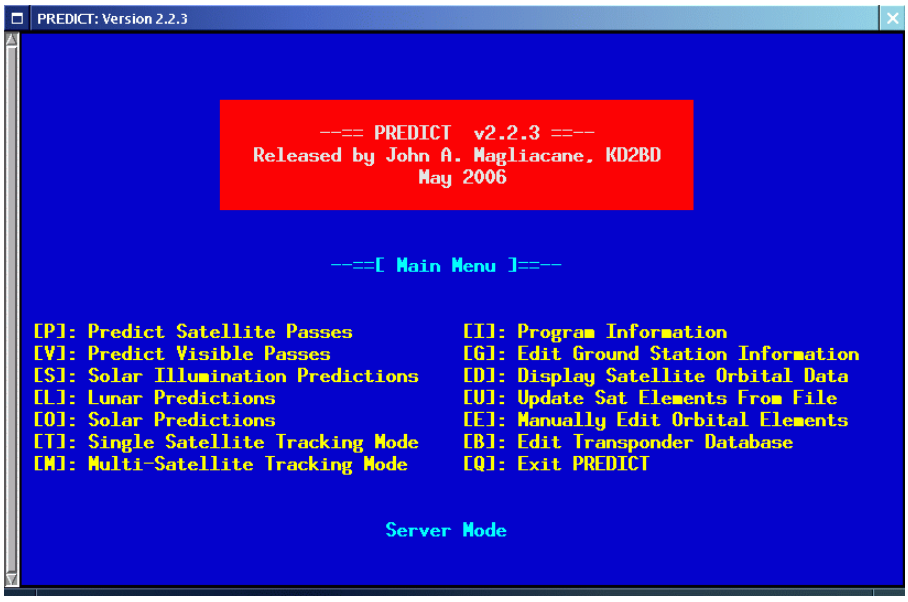


Figura 6. Ventana principal de PREDICT.

Para mayor comodidad del usuario este software se distribuye preparado para varias plataformas:

- Preparado para sistemas UNIX o Linux en forma de código fuente.

- Preparado para sistemas DOS en forma de paquetes sin compilar y paquetes precompilados listos para su uso.
- Precompilado dentro de la distribución Portable PREDICT Plus!

Podemos comprobar en su sitio web que no existe una versión específica para Mac OS X. Tendremos que realizar unas modificaciones al código fuente existente para los sistemas de tipo Linux para obtener así una versión compatible.

Acompañando a esta aplicación se han creado multitud de desarrollos que complementan sus puntos débiles.
Podemos clasificar estos desarrollos en tres grupos:

- Rutinas para mejorar el seguimiento automatizado permitiendo la conexión a una mayor variedad de sistemas de rotores.
    - PIC/TRACK Antenna Rotator Controller.
    - EasyTrak
    - FODTRACK rotator interface

- Rutinas para permitir la conexión remota a las estaciones.
    - JCP.
    - The g7III PREDICT Clients.
    - PCRSAT.
    - SYNOP.

- Desarrollos completos con una interfaz gráfica que utilizan como partida PREDICT. Los más importantes son:
    - Gpredict.
    - ISS Spotter.
    - PetitTrack.
    - Ktrack.
    - APTDecoder.

  En este mismo apartado detallaremos las características y las funcionalidades de dos aplicaciones de la lista anterior, Gpredict y PetitTrack.

Este programa se podrá obtener desde la web de su autor (John A. Magliacane, 2013)[13]. Este desarrollo se encuentra bajo una licencia del tipo GNU General License GPL (Free Software Foundation, 2013)[14].

Portable PREDICT Plus!

Portable PREDICT Plus! es, más que una aplicación, una minidistribución de Linux basada en el kernel 2.6.9 pensada para ejecutarse directamente en la memoria RAM.

[13] http://www.qsl.net/kd2bd/predict.html
[14] http://www.gnu.org/licenses/gpl.html

Esta distro está optimizada para ocupar solo dos diskettes de 3'5 pulgadas y funcionar en equipos con muy bajos recursos.

Esta distribución contiene las siguientes aplicaciones:

- PREDICT. La aplicación anteriormente mencionada, eso sí, con algunas limitaciones en sus funcionalidades.
- MoonTracker.
- FodTrack. Utilidad para manejar la altura y acimut de los rotores. También sirve para controlar un transreceptor[15].
- PB/PG
- AX.25 Utils. Utilidades para manejar comunicaciones con el protocolo AX25[16].
- Minicom. Programa para realizar las comunicaciones seriales del dispositivo.
- PacsatTools. Utilidades para facilitar las comunicaciones con satélites del tipo PacSat[17].

Aunque está distribución se encuentra actualmente algo desfasada, puede ser útil en según qué entornos.

Gpredict.

Gpredict se puede considerar la aplicación derivada de PREDICT más popular.
Partiendo de PREDICT se desarrolló una interfaz gráfica con GTK para facilitar su uso.

Las funcionalidades de este software, según su página oficial, son las siguientes[18]:

- Seguimiento de satélites atendiendo a los algoritmos SGP4/SDP4 del NORAD.
- Sin limitaciones por software en cuanto a los satélites a seguir o a las estaciones terrestres en uso.
- Posibilidad de mostrar la información en mapas, tablas o diagramas polares.
- Posibilidad de organizar la información por módulos.
- Control automático de rotores y de sistemas de radio.
- Posibilidad de configurar completamente la apariencia gráfica de la interfaz.
- Actualización automática de los elementos TLE mediante HTTP, FTP o mediante descarga manual.

Este desarrollo está disponible, en paquetes precompilados, para los sistemas operativos más comunes. Se ofrece también la posibilidad de descargar el código fuente desde la página web personal del desarrollador en el sitio web Sourceforge[19].

---

[15] http://www.qsl.net/ve2dx/projects/fod.htm
[16] http://www.linux-ax25.org/wiki/LinuxAX25
[17] http://www.qsl.net/kd2bd/software.html
[18] http://gpredict.oz9aec.net/features.php
[19] https://sourceforge.net/projects/gpredict/files/Gpredict/

El código de este proyecto se encuentra licenciado, como es habitual, bajo una licencia del tipo GNU General Public License (Free Software Foundation, 2013)[20].

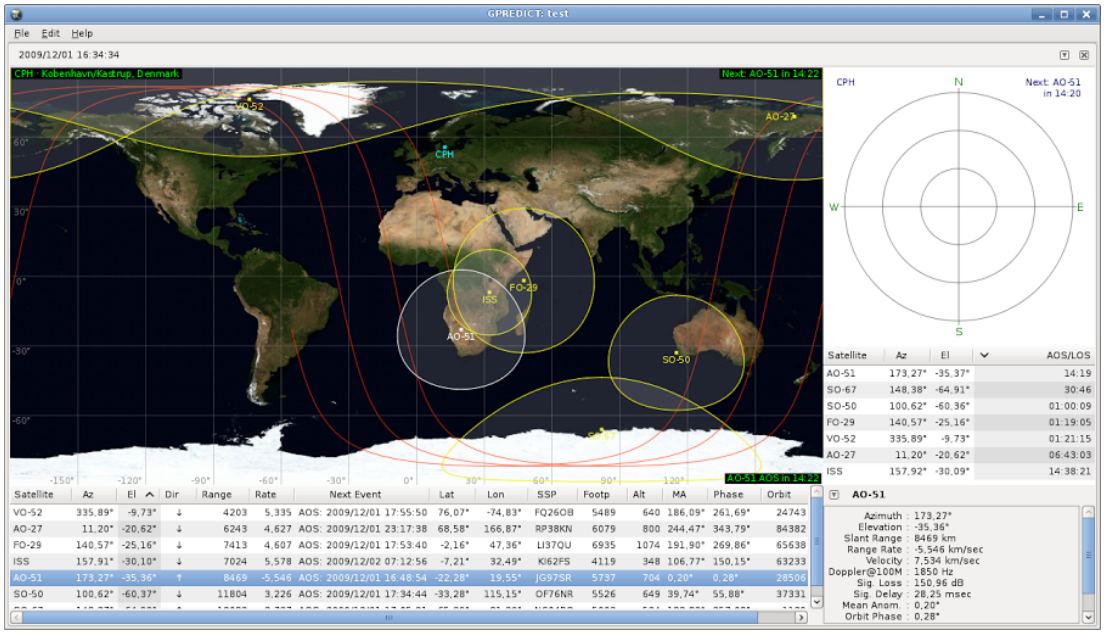


Figura 7. Ventana principal de la aplicación en Linux (Csete)[21].

Aunque esta aplicación está pensada para ser altamente configurable, presenta la tara de ser muy difícilmente adaptable para su uso en pantallas con una diagonal inferior a siete pulgadas. Esto nos imposibilita su utilización en prototipos con pequeñas pantallas.

Tampoco podremos hacer uso de displays leds y botones de selección para interactuar con el programa, pues no está preparado para recibir o transmitir información por este método.

PetitTrack.

Al igual que JSatTrack, PetitTrack no permite el guiado de rotores. Lo mencionaremos en este punto porque es el único desarrollo que permite su ejecución en sistemas embebidos.

---

[20] http://www.gnu.org/licenses/gpl.html
[21] Captura de pantalla obtenida del perfil de G+ del desarrollador.

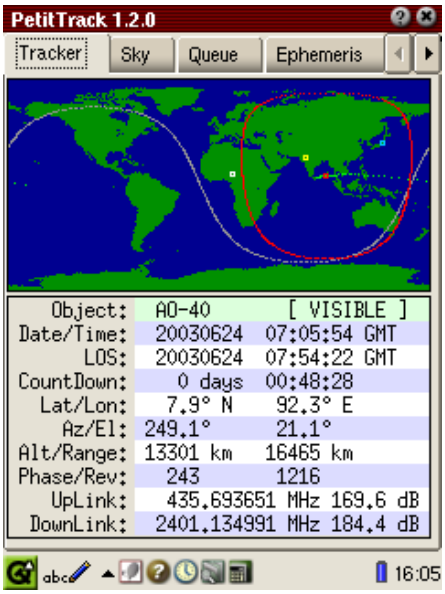


Figura 8. Ventana principal de PetitTrack.

Ya que PetitTrack nació inspirado por PREDICT sus funcionalidades serán similares a este último, permitiéndonos:

- Mostrar gráficos en 2D con la posición actual del objeto.
- Mostrar la posición de la Luna y el Sol.
- Mostrar las efemérides del objeto en tiempo real y para un tiempo dado.
- Definir diversas posiciones para el observador.

La descarga de la aplicación se podrá realizar desde la web del desarrollador[22].
Las instrucciones de instalación se encuentran detalladas en esa misma página web.

Librerías.

Ya que vamos a utilizar el lenguaje de programación Python, enumeraremos una lista de librerías creadas con esta plataforma. Si nombráramos también las librerías existentes en otros lenguajes de programación obtendríamos una lista demasiado extensa, y este no es el objeto de esta memoria.

PyEphem.

PyEphem[23] consiste en un módulo para el lenguaje de programación Python creado partiendo de las librerías de la aplicación de carácter astronómico XEphem[24].

[22] http://www.qsl.net/n1vtn/petittrack.html
[23] http://rhodesmill.org/pyephem/
[24] http://www.clearskyinstitute.com/xephem/

Heredando la licencia de XEphem, PyEphem está publicado bajo una licencia del tipo LGPL (Free Software Fundation, 2013)[25].

Este módulo contiene implementaciones para realizar cálculos de tipo astronómico.
Si se le proporciona una fecha y una posición en la superficie terrestre, PyEphem puede calcular la posición del Sol y de la Luna, de los planetas y sus satélites, además de la situación en la esfera celestre de cualquier cuerpo menor, natural o artificial, del Sistema Solar.

También puede calcular salidas, tránsitos y ortos de objetos además de distancias angulares en la bóveda celeste.

El funcionamiento de este módulo será el mismo que el de cualquier otra librería de Python. Pondremos un ejemplo extraído de la web del desarrollador[26].

```
1  >>> gatech = ephem.Observer()
2  >>> gatech.lon, gatech.lat = '-84.39733', '33.775867'
3  >>> gatech.date = '1984/5/30 16:22:56'   # 12:22:56 EDT
4  >>> sun, moon = ephem.Sun(), ephem.Moon()
5  >>> sun.compute(gatech)
6  >>> moon.compute(gatech)
7  >>> print sun.alt, sun.az
8  70:08:39.2 122:11:26.4
9  >>> print moon.alt, moon.az
10 70:08:39.5 122:11:26.0
```

El creador de la librería nos da, en esa página, algunos ejemplos de utilización de su módulo mediante el intérprete de Python.
En este ejemplo hemos numerado las líneas para facilitar su comprensión.

En la primera línea crearemos un objeto de la clase *Observer* que pertenecerá al módulo *ephem*.
Una vez creado el objeto podremos acceder a los atributos públicos de éste. En este caso modificaremos sus atributos de longitud y latitud. Esto se hará, en la segunda línea, por medio de sus atributos *lon* y *lat*.
La fecha y hora se fijará por medio del atributo *date* en la tercera línea.

Para obtener información sobre un cuerpo celeste tendremos que crear un objeto de su clase correspondiente. Para los cuerpos celestes más importantes existen clases específicas. Así, para el Sol y la Luna, existen las clases *Sun* y *Moon,* al igual que cada planeta tiene su clase correspondiente: *Mercury, Venus, Mars…*
En la cuarta línea se crean dos objetos de la clase *Sun* y de la clase *Moon*.

Para obtener sus efemérides tendremos que invocar al método público, *compute,* de la clase del objeto a observar dándole como argumento el objeto que define la ubicación. En

[25] http://www.gnu.org/licenses/lgpl.html
[26] http://rhodesmill.org/pyephem/tutorial.html#computations-for-particular-observers

nuestro caso le daremos como argumento "gatech", nuestra ubicación definida en las líneas segunda y tercera.

Una vez invocada la función *compute* ya tendremos disponibles las efemérides del objeto. Conseguir estas efemérides será tan sencillo como llamar a los atributos del objeto.

En nuestro caso solo hemos solicitado el valor de la altura (*alt*) y del acimut (*az*) pero tenemos disponibles una larga lista de valores. La mayoría de ellos serán los siguientes[27]:

- Para todos los objetos.
    - *a_ra.* Ascensión recta astrométrica desde una posición geocéntrica para la época especificada[28].
    - *a_dec.* Declinación astrométrica desde una posición geocéntrica para la época especificada.
    - *g_ra.* Ascensión recta aparente desde una posición geocéntrica para la época de la observación
    - *g_dec.* Declinación aparente desde una posición geocéntrica para la época de la observación.
    - *ra.* Ascensión recta aparente desde una posición topocéntrica para la época de la observación.
    - *dec.* Declinación aparente desde una posición topocéntrica para la época de la observación.
    - *elong.* Elongación del objeto.
    - *mag.* Magnitud aparente del objeto.
    - *size.* Diámetro del objeto en segundos de arco.
    - *radius.* Tamaño visual del radio del objeto en grados.
- Para los objetos del sistema solar además tendremos los siguientes valores.
    - *hlon.* Longitud heliocéntrica.
    - *hlat.* Latitud heliocéntrica.
    - *sun_distance.* Distancia al Sol medida en unidades astronómicas.
    - *earth_distance.* Distancia a la Tierra expresada también en unidades astronómicas.
    - *phase.* Fase. Porcentaje de la superficie del objeto iluminada por el Sol.
- Para las lunas de los planetas del sistema solar también estarán disponibles.

  Posición relativa de las lunas con respecto al planeta que orbitan medida en radios de este.
    - *x.* Posición medida con respecto al este u oeste. Siendo el este un valor positivo y el oeste un valor negativo.
    - *y.* Posición medida con respecto al sur o norte. Los valores positivos expresarán valores al sur del centro del planeta representado los valores negativos posiciones al norte de este.
    - *z.* Con respecto a la Tierra, *z* expresará si se encuentra delante o detrás del planeta desde nuestro punto de vista. Se considerará que los valores

[27] http://rhodesmill.org/pyephem/quick.html#body-compute-date
[28] http://rhodesmill.org/pyephem/radec

negativos expresan que el objeto se encuentra "detrás" del planeta expresando valores positivos que el satélite se encuentra entre el planeta y la tierra.
    - *earth_visible.* Visibilidad de la Luna desde la Tierra.
    - *sun_visible.* Visibilidad de la Luna esta vez desde el Sol.
- Si el objeto es un satélite artificial podremos pedir también.
    - *sublat.* Latitud del punto de la superficie terrestre sobre el cual se halla el objeto. Latitudes septentrionales son positivas.
    - *sublong.* Longitud del punto de la superficie terrestre sobre el que se encuentra el satélite. Las longitudes orientales son positivas.
    - *elevation.* Elevación del satélite sobre la superficie de la Tierra.
    - *range.* Distancia del observador al satélite.
    - *range_velocity.* Velocidad a la cual cambia la distancia del observador al satélite. Este cambio se mide en metros por segundo.
    - *eclipsed.* Parámetro que nos indica si el objeto se encuentra eclipsado por la sombra de la Tierra o no.
- La Luna también dispondrá de los siguientes atributos (Wikipedia, 2013)[29].
    - *libration_lat.* Latitud de la libración lunar actual.
    - *libration_long.* Longitud de la libración lunar en este instante.
    - *colong.* Colongitud selenográfica.
    - *moon_phase.* Porcentaje de la Luna iluminada.
    - *subsolar_lat.* Latitud lunar sobre la que se encuentra el Sol.
- Para los satélites jovianos tendremos.
    - *cmll.* Longitud del meridiano central en el sistema I.
    - *cmlll.* Longitud del meridiano central esta vez según el sistema II.
- Por último, para los cuerpos pertenecientes al sistema Saturniano.
    - *earth_tilt.* Inclinación de los anillos de Saturno vistos desde la Tierra.
    - *sun_tilt.* Inclinación de los anillos esta vez vistos desde la posición del Sol.

La documentación del módulo se podrá obtener de la web del desarrollador. En este mismo sitio web se encuentran instrucciones detalladas del proceso de instalación del módulo.
Existe un repositorio[30] en GitHub del creador de la librería donde se puede obtener la última versión de este proyecto.

Astropy.

Según la introducción de su sitio web[31] *"El proyecto Astropy es un esfuerzo comunitario para desarrollar un módulo para astronomía en Python y fomentar la interoperabilidad entre los paquetes de Python de contenido astronómico".*

[29] http://es.wikipedia.org/wiki/Coordenadas_selenográficas
[30] https://github.com/brandon-rhodes/pyephem
[31] http://www.astropy.org/

Acorde a esta descripción este módulo nos facilita una serie de funciones de muy diversa índole, eso sí, todas ellas relacionadas con el campo de la astronomía.
Ya que nuestro interés reside en saber qué nos ofrece cada librería, pasaremos a detallar qué funcionalidades nos proporciona este paquete[32].

- Estructuras de datos y transformaciones de éstos.
    - Constantes. El subpaquete *constants* reúne un conjunto de constantes físicas útiles en cálculos astronómicos.
    - Unidades y cantidades. Con el subpaquete *units* podremos realizar conversiones entre valores expresados en diferentes unidades y cálculos aritméticos con estos datos.
    - Análisis N-dimensional. Utilizando el subpaquete *nddata* tendremos acceso a la clase NDData que, junto a sus herramientas, nos dará la posibilidad de realizar cálculos con arrays multidimensionales.
    - Tablas de datos. *table* nos ayudará con la tarea de manejar tablas de valores con una forma de trabajo similar al paquete *numpy*[33].
    - Fechas y tiempos. El paquete *time* nos dará la habilidad de poder manejar diferentes formas de fechas y horas. Para incrementar su eficiencia, todo el código de este subpaquete está programado mediante Cython[34].
    - Sistemas de coordenadas astronómicas. El subpaquete *coordinates* nos proporciona utilidades para las conversiones de datos entre diferentes sistemas de coordenadas.
    - Sistema mundial de coordenadas. Las rutinas de mapeo de archivos fits estarán proporcionadas por el subpaquete *wcs*. Este paquete nos permitirá aplicar transformaciones del tipo wcs (world coordinated system) a nuestras imágenes para relacionar cada pixel del archivo fit con una coordenada celeste real.
    - Modelos y ajustes. Mediante el subpaquete *modeling* podremos realizar ajustes y modelos en una y dos dimensiones partiendo de los datos que le proporcionemos.
- Entrada y salida de datos.
    - Interfaz unificada de lectura y escritura de datos. Astropy intenta ofrecer una interfaz unificada para la lectura y escritura de datos de diferentes tipos. Según su sitio web oficial, la cantidad de formatos soportados aumentará ya que esta funcionalidad se encuentra actualmente en desarrollo.
    - Manejo de ficheros FITS. El paquete *io.fits* perteneciente a *Astropy* nos proporciona la habilidad de manejar ficheros del tipo fits.
    - Tablas ASCII. Las tablas de datos en formato ASCII son accesibles mediante el subpaquete *io.ascii*.
    - Manejo de archivos XML con formato VOTable. El paquete *io.votable* convierte archivos XML con formato VOTable a arrays de datos con formato Numpy y viceversa.

[32] http://docs.astropy.org/en/stable/#user-documentation
[33] http://www.numpy.org/
[34] http://cython.org/

    - Funciones varias de entrada y salida. Las funciones que no se pueden clasificar en los tres puntos anteriores se agrupan en el módulo *io.misc*.
- Utilidades de astronomía computacional.
    - Filtrado y convolución de imágenes. El paquete convolution provee las rutinas necesarias para los tratamientos de filtrado y convolución de archivos fits.
    - Cálculos cosmológicos. Las clases encargadas de la realización de cálculos de tipo cosmológico se encuentra en el paquete *cosmology.* Este paquete nos proporciona las funciones necesarias para los cálculos dependientes de modelos cosmológicos. Dentro de estos podemos encontrar entre otros:
        - Cálculos de distancias y edades mediante desplazamiento al rojo.
        - Cálculos de distancias mediante medidas angulares.
    - Herramientas astroestadísticas. El paquete *stats* reúne diversas funciones estadísticas y algoritmos usados en astronomía.
    - Acceso al observatorio virtual. El IVOA[35] (acrónimo de Alianza Internacional del observatorio virtual) es una organización que suministra libre y gratuitamente catálogos estelares, mediante su API, a cualquiera que lo necesite. Astropy posee las utilidades necesarias, mediante su subpaquete *vo* para acceder a estos recursos.
- Utilidades varias de la librería.
    - Configuración del sistema. Para facilitar la configuración del sistema Astropy reúne todos los parámetros de configuración del sistema en el paquete *config*. Esto evita que el usuario tenga que modificar el código fuente del desarrollo para adaptarlo a sus intereses.
    - Registo E/S. El submódulo *io.registry* tiene la finalidad de permitir al usuario modificar la forma con la que la librería interactúa con los archivos que entran y salen del sistema.
    - Sistema de registro. El sistema de registro de Astropy (logging system) permite la configuración de qué mensajes de registro se mostrarán y cuáles no. Para ello tendremos que importar la clase denominada *log*.
    - Sistema de alertas. El sistema de alertas del módulo Astropy hereda las características del sistema de Python. Esto no quita que las alertas se puedan suprimir si así se desea.
    - Utilidades del paquete Astropy. Dentro del subpaquete *utils* se engloban todas aquellas utilidades que no entran en ninguno de los grupos anteriores. Algunas de las funcionalidades que nos otorga *utils* son:
        - Control de excepciones.
        - Control del módulo por consola.
        - Control del tiempo.
        - Descarga de ficheros.
        - Comprobación del estado de ficheros del tipo XML.

Como podemos comprobar, Astropy es una librería mucho más completa que PyEphem.

[35] http://www.ivoa.net/

En su favor también tendremos que decir que cuenta con una comunidad de desarrollo muy activa. La última versión de este proyecto, la 0.3, salió a la luz en noviembre de 2013.

En su página web principal encontraremos el proceso de instalación para OS X, Linux y Windows. Si deseamos obtener el código fuente de este proyecto tendremos que visitar el repositorio oficial de éste[36] en GitHub.
La licencia de este proyecto no será del tipo GNU-GPL, sino que será una licencia de clase BSD (Open Source Initiative)[37]. Estas licencias son menos restrictivas sobre el uso que se puede dar al código licenciado y no tendría por qué haber ningún problema con usar este proyecto en cualquier desarrollo.

stsci_python.

Este módulo consiste en otra recopilación de subpaquetes destinados al análisis y cálculo de variables astronómicas. Conformada casi en su totalidad por rutinas escritas en Python también posee extensiones en C en algunas de sus componentes.
La última versión, lanzada en mayo de 2013 contiene los siguientes módulos[38]:

- PyRAF. Entorno desarrollado para poder utilizar IRAF sin tener que usar el entorno de usuario por defecto del sistema. La última versión liberada de este desarrollo es la número 2.1.5.
- MultiDrizzle. Utilidad destinada a crear mosaicos de imágenes obtenidas con el Telescopio Espacil Hubble. Este módulo nos otorga también la oportunidad de obtener datos astrómetricos de nuestras creaciones.
- PyFITS. Modulo que nos proporciona la funcionalidad para leer y modificar imágenes con formato FITS.
- pysynphot. Paquete en desarrollo destinado a los estudios de carácter fotométrico.
- Numdisplay. Módulo que nos permite visualizar arrays de datos usando utilidades para mostrar imágenes como ds9[39] y ximtool[40].

El código de este proyecto no se encontrará, al contrario que en los desarrollos anteriores, alojado en GitHub.
Para facilitar su instalación los desarrolladores del proyecto han alojado su código en PyPI. Podremos instalar el desarrollo usando pip o descargando el código desde la web[41] y compilándolo manualmente.
Este módulo se encuentra bajo una licencia del tipo BSD, así que tampoco habrá ningún problema para usarlo en nuestro desarrollo.

---

[36] https://github.com/astropy/astropy
[37] http://opensource.org/licenses/BSD-3-Clause
[38] http://www.stsci.edu/institute/software_hardware/pyraf/stsci_python
[39] http://hea-www.harvard.edu/RD/ds9/site/Home.html
[40] ftp://iraf.noao.edu/iraf/x11iraf/
[41] https://pypi.python.org/pypi/stscipython/2.14

Astrolib.

Astrolib es posiblemente una de las compilaciones más en desuso actualmente.
De los siete módulos que incorpora, cuatro ya pertenecen a otras distribuciones y uno de ellos tiene su desarrollo paralizado.
Actualmente la compilación se compone de los siguientes elementos:

- AstroAsciiData. Módulo para la lectura, modificación y escritura de tablas de datos en formato ASCII.
- Asciitable. Con este módulo incrementaremos las posibilidades de lectura y escritura de tablas ASCII.
- Pysynphot.
- coords.
- tfit. Módulo para el análisis fotométrico de archivos con diferentes resoluciones. La última versión fue liberada en julio de 2008, actualmente se considera un proyecto inactivo.
- pywcs.
- vo.table.

Las licencias serán diferentes para cada uno de los desarrollos. Tendremos que irnos a la página web de cada uno de ellos para consultarlas.

APLpy.

El propósito principal de este módulo es la creación de imágenes astronómicas en formato FITS.
Acorde a su sitio web[42], APLpy tendrá las siguientes funcionalidades:

- La creación de mapas de una manera interactiva o mediante el uso de scripts.
- La posibilidad de variar la escala de colores de esos mapas mostrándolos de tres maneras distintas: en blanco y negro, en color y descompuestos en tres canales de colores (rojo, verde y azul).
- La posibilidad de unir los contornos de imágenes para la creación de imágenes mayores.
- El añadido de escalas y marcadores a los mapas creados.
- La posibilidad de añadir controles para hacer zoom y desplazamientos por las imágenes creadas.
- La opción de guardar los archivos obtenidos en diversos formatos, como son: EPS, PDF, PS, PNG y SVG.

Este módulo tendrá el inconveniente de necesitar para su funcionamiento la instalación de otros tres paquetes: Numpy, Matplotlib y Astropy.
Para instalar este módulo podemos descargar el código del proyecto desde su repositorio[43] oficial en GitHub.

---

[42] http://aplpy.github.io/
[43] https://github.com/aplpy/aplpy

Al contrario que las librerías mencionadas anteriormente, este módulo no posee ninguna licencia. Esto quiere decir que al usarlo tendremos que atenernos a las licencias heredadas de los paquetes de los que depende.

Proyectos de hardware libre actuales.

Como soporte del sofware anteriormente mencionado tenemos una amplia variedad de placas de desarrollo.
La mayor parte de estas placas de desarrollo tienen parte, o todo, de sus componentes y diseños bajo una licencia libre. Esto quiere decir que se podrán utilizar en desarrollos comerciales sin ningún tipo de problema.
En este punto mencionaremos las más interesantes, tanto por su comunidad de desarrolladores como por su relación calidad-precio.
Cuando analicemos los pros y contras de cada una de ellas en el siguiente capítulo analizaremos los tipos de licencias de que poseen.

Raspberry Pi.

La idea de la creación de esta placa de prototipado nace en el año 2006 en el laboratorio de computación de la Universidad de Cambridge.
Algunos de sus miembros, preocupados por el desinterés en estas temáticas, comenzaron el proceso de desarrollo de un dispositivo que recuperara el espíritu de los viejos microordenadores.

Los antiguos microordenadores habían sido sustituidos por ordenadores mucho más caros. Dispositivos cuyas habilidades para el desarrollo de prototipos eran mucho más limitadas. Dispositivos demasiado valiosos para utilizarlos en la experimentación.

Después de varios diseños iniciales, en el año 2008, se decidió utilizar un microprocesador de teléfono móvil. Los teléfonos inteligentes habían irrumpido en el mercado provocando un brusco descenso de los precios. El modelo actual, fechado en el año 2012, cuenta con uno de ellos.

Este modelo está basado en un chip Broadcomm BCM2835 [44], que incluye el microprocesador, la GPU, la RAM y el sistema de control de puertos, y un integrado LAN9512[45] para gestionar sus dos puertos USB y su puerto Ethernet. Este integrado le permite ejecutar con relatividad comodidad un entorno de escritorio completo. En la actualidad, el listado de sistemas operativos disponibles para este microordenador incluye algunas de las distribuciones más conocidas basadas en el kernel de Linux además de una versión optimizada de RISC OS.

---

[44] http://www.raspberrypi.org/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf
[45] http://ww1.microchip.com/downloads/en/DeviceDoc/9512db.pdf

El sistema operativo por antonomasia que utiliza esta placa es Raspbian, versión de Debian muy optimizada para este sistema. En la actualidad está basada en la última versión de Debian disponible (7.0).

También está disponible un modelo con características inferiores, llamado "Model A", destinado a tareas que requieran menos recursos. Este modelo dispone de una cantidad inferior de memoria RAM, 256 megabytes, además de solo contar con un puerto USB y no disponer de puerto Ethernet.

Aunque ambos modelos, "Model A" y "Model B", poseen algunas diferencias significativas, la variedad de conexiones disponibles en ellos y la enorme comunidad de desarrolladores que utilizan éstas, facilita la creación de software para estos dispositivos.

Figura 9. Raspberry Pi (Wikipedia Commons, 2012) [46].

Ya que este es un proyecto educativo y se prima la facilidad de uso, y no la potencia de cálculo, no existen previsiones de una gran actualización de estos dispositivos en un futuro cercano.
El coste de estas placas según su página oficial es de 25 dólares para el modelo A y de 35 dólares para el modelo B sin contar impuestos y gastos de envío.

Cubieboard.

La placa de desarrollo Cubieboard[47] pertenece a la familia de dispositivos nacidos a raíz del nacimiento de Raspberry Pi.
El diseño, producción y comercialización de esta placa corre a cargo de Cubietech Limited, empresa radicada en Zhuhai, China. Dentro de esta empresa, el autodenominado

[46] http://upload.wikimedia.org/wikipedia/commons/3/3d/RaspberryPi.jpg
[47] http://cubieboard.org/

Cubieteam ha creado tres modelos de este desarrollo: Cubieboard, Cubieboard2 y Cubietruck (también llamado Cubieboard3).

Aunque con una comunidad de desarrolladores bastante menor, existe un foro[48] oficial, una lista de correo y una wiki[49] mantenidos por la compañía que desarrolla este dispositivo.
Esta comunidad ha adaptado algunos de los sistemas operativos basados en Linux más populares, los cuales son:

- Arch Linux.
- Berryboot.
- Debian, usando el entorno de escritorio LXDE.
- Fedora.
- Kali.
- Mer.
- Tiny Core.
- Lubuntu.
- Android. La versión de Android disponible variará según la placa elegida.

Éstos se encuentran disponibles para su descarga desde la web oficial de Cubietech Limited. A continuación pasaremos a detallar las características de cada modelo.

Cubieboard y Cubieboard2.

La diferencia entre la placa Cubieboard y la placa Cubieboard2 será el SoC utilizado. La disposición de los elementos y el esquema de los pines será el mismo en ambos modelos. Esto permitirá usar los mismos accesorios en ambas placas.

Así mientras la placa Cubieboard cuenta con un SOC A10 con un microprocesador ARM® CORTEX™ A8 de un núcleo, la placa Cubieboard2 cuenta con un SOC A20 que contiene un microprocesador ARM® CORTEX™ A7 de doble núcleo. El resto de características serán:

- 1 gigabyte de memoria RAM DDR3 a 480 megahertzios.
- Alrededor de 4 gigabytes de memoria integrada en placa, con la posibilidad de usar tarjetas SD de hasta 64 gigabytes o discos duros de hasta 2 terabytes.
- Ethernet 10/100 Mbps con capacidad de usar Wi-Fi mediante adaptadores USB.
- Conexión HDMI con capacidad 1080p.
- Conector SATA para discos duros de 2’5 pulgadas.
- 2 puertos USB HOST. 1 puerto USB OTG.
- Una entrada infrarroja. Entrada y salida de audio.
- 96 puertos para comunicaciones. Incluyendo: I2C, SPI, RGB/LVDS, CSI/TS, FM-IN, ADC, CVBS, VGA, SPDIF-OUT, R-TP…

[48] http://www.cubieforums.com/
[49] http://docs.cubieboard.org/

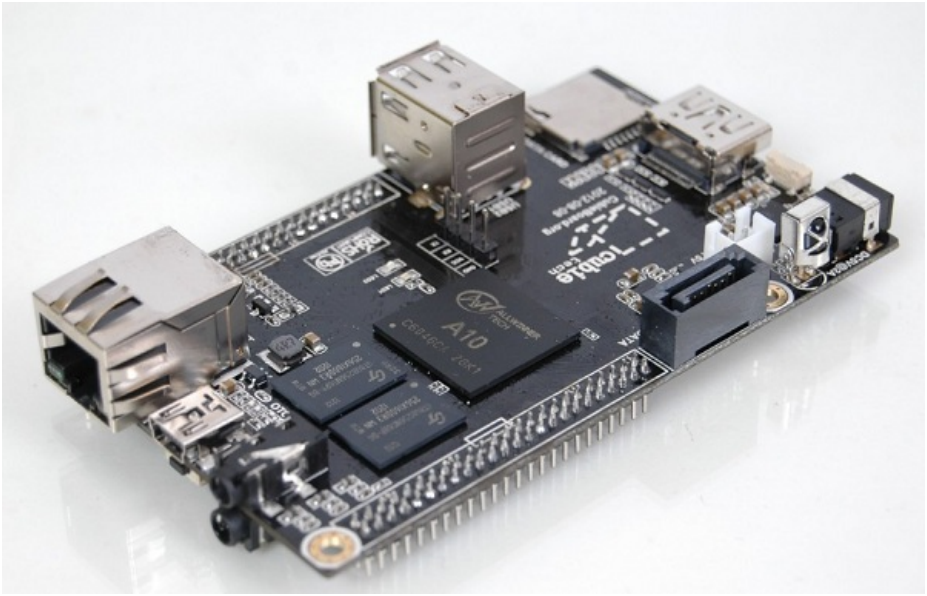


Figura 10. Cubieboard (Cubietech Ltd.) [50].

Cubietruck (o Cubieboard3).

El diseño de la placa Cubietruck será muy diferente al de las dos placas anteriores. Enfocada a un mercado más profesional la cantidad de puertos y conexiones disponibles será mucho mayor en este caso.

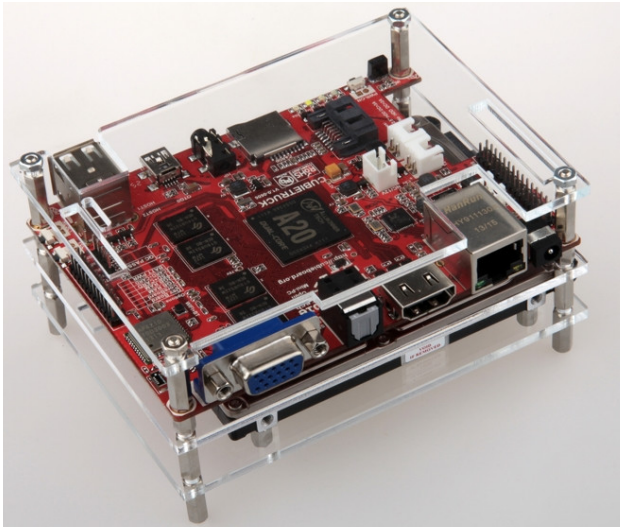


Figura 11. Cubietruck (Cubietech Ltd.) [51].

Las características de este modelo serán las siguientes:

- AllWinnerTech SOC A20 que cuenta con un microprocesador ARM® Cortex™ A7 de doble núcleo.

[50] http://docs.cubieboard.org/_media/cubieboard.jpg
[51] http://docs.cubieboard.org/_media/a20-cubietruck.png

- Posibilidad de elegir entre un modelo con 1 o 2 gigabytes de memoria RAM.
- Ethernet 10/100/1000 Mbps.
- Conexión Wi-Fi y Bluetooth con antena propia integrada en placa.
- Salida a pantalla mediante puerto HDMI o VGA.
- Reloj de tiempo real (RTC).
- Diferentes configuraciones de almacenamiento basado en tarjetas micro-SD y memoria integrada en placa.
- Interfaz SATA 2.0 con alimentación en placa para discos de 2'5 pulgadas. Da la opción de usar un alimentador externo para el uso de discos de 3'5 pulgadas.
- Controlador de batería externa.
- 54 puertos para comunicaciones. Incluyendo: I2S, I2C, SPI, CVBS, LRADC, UART, PS2, PWM, TS/CSI, IRDA, LINEIN, FMIN, MICIN y TVIN.

El precio de estas placas se encontrará en un margen de precios superior al de los dos modelos de la Raspberry Pi Foundation mencionados anteriormente.

Así, el precio medio de la Cubieboard en una tienda española será de 65 euros, excluyendo gastos de envío. La Cubieboard2 rondará los 80 euros costando la Cubieboard3 un promedio de 120 euros.

Beagleboard.

Nacidas, igualmente, al amparo del boom de los microordenadores, las placas de prototipado Beagleboard representan una opción también interesante.

Estas placas son creadas y comercializadas por la fundación sin ánimo de lucro estadounidense *BeagleBoard.org Foundation*, que pretende con ellas promover el uso del hardware (y software) libre en entornos de computación embebida.
Esta fundación intenta estrechar lazos entre desarrolladores mediante un foro hospedado en su página[52] y un chat al que se puede acceder desde ésta.
También se ofrece una lista de correo para los entusiastas de este campo.

Los precios de sus desarrollos están orientados a estudiantes y a entornos no profesionales.

Los modelos disponibles actualmente son los siguientes:

- BeagleBone Black.
- BeagleBone.
- BeagleBoard-xM.
- BeagleBoard.

[52] http://beagleboard.org/

En éstos puedes ejecutar una variada selección de sistemas operativos. Los desarrollos oficiales son los siguientes:

- Basados en el kernel de Linux:
    - Angstrom.
    - Ubuntu.
    - Debian.
    - ArchLinux.
    - Gentoo.
    - Sabayon.
    - Buildroot.
    - Erlang.
    - Fedora.

- Desarrollados por Texas Instrument:
    - Android.
    - Linux.

- Basados en otros núcleos:
    - QNX.
    - FreeBSD.

La *BeagleBoard Foundation* también ofrece, a través de Texas Instrument, un conjunto de librerías llamado StarterWare para su uso en sistemas de tiempo real (RTOS).

A continuación, como es habitual, describiremos las funcionalidades de cada modelo. Realicemos nuestro recorrido por orden cronológico.

BeagleBoard.

El lanzamiento de este microordenador se produjo el 28 de julio de 2008, siendo así el primer desarrollo de esta fundación.

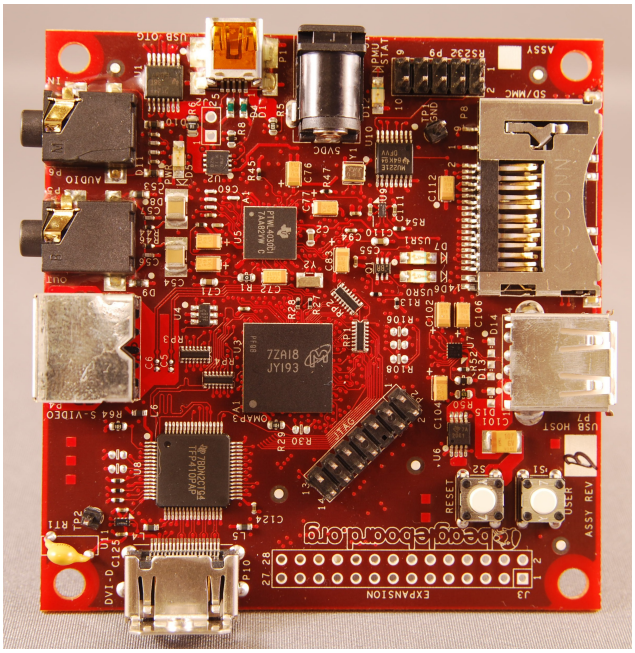


Figura 12. BeagleBoard (Kridner, 2008) [53].

Esta placa, para mejorar su productividad, ha sufrido diferentes actualizaciones desde su fecha de lanzamiento.
La versión actual, la D, tiene las siguientes características técnicas:

- Texas Instrument SOC OMAP3530 que cuenta con un microprocesador ARM® CORTEX™A8 a 720 megahercios.
- 256 megabytes de memoria RAM de bajo consumo.
- 512 megabytes de memoria en placa.
- Posibilidad de alimentación mediante un puerto USB OTG o mediante un conector de tipo cilíndrico.
- Interfaz DVD-D e interfaz S-Video para la conexión a monitores y televisiones.
- Conector serie RS232 en placa.
- Conector JTAG para la depuración del sistema.
- Interfaz Ethernet 10/100.
- Conector para entrada de audio estéreo.
- Conector para salida de audio estéreo.
- Puerto USB para conectar periféricos de todo tipo.
- Posibilidad de utilizar como almacenamiento externo tarjetas de memoria de formato SD y MMC.
- 28 pines de comunicaciones que incluyen conexiones de tipo: I2C, GPIO, PWM, MMC, UART, McBSP, MsSPI.

Al ser un proyecto de hardware libre todos los elementos del diseño de esta placa de prototipado serán accesibles al gran público. El esquemático del circuito y el diseño de la placa, además de la lista de materiales empleados, se podrán descargar desde la wiki del fabricante[53].

BeagleBoard-xM.

Concebida como una renovación de la BeagleBoard, esta placa es una actualización de los componentes del anterior modelo.
Esta renovación intenta ampliar las funcionalidades del dispositivo anterior.

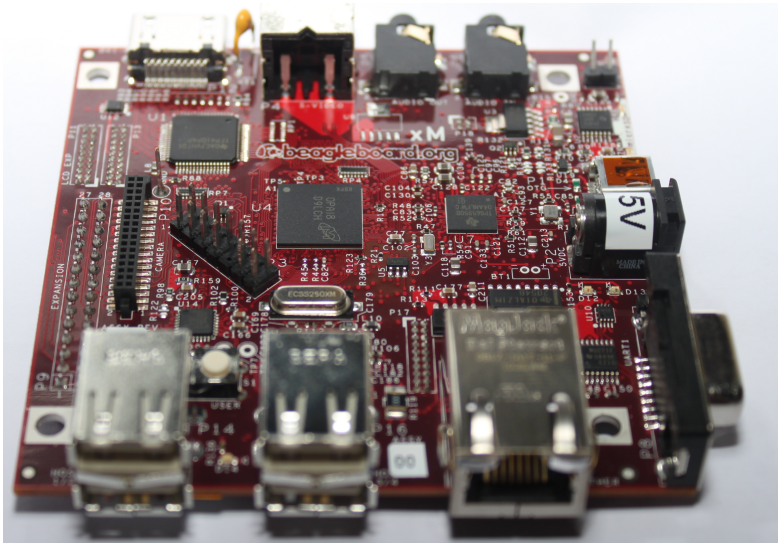


Figura 13. BeagleBoard-xM (Wikipedia Commons, 2011)[54].

La lista de funcionalidades ampliadas del modelo actual (revisión C2) se puede resumir en los siguientes puntos:

- Texas Instrument SOC AM37x con un microprocesador ARM® CORTEX™ A8 con una frecuencia de un gigahercio.
- 512 megabytes de memoria RAM DDR2 de bajo consumo.
- Posibilidad de alimentación mediante un puerto USB OTG o mediante un conector de tipo cilíndrico.
- Interfaz DVD-D e interfaz S-Video para la conexión a monitores y televisiones.
- Conector JTAG para la depuración del sistema.
- Interfaz Ethernet 10/100.
- Conector para la entrada de audio estéreo.
- Conector para la salida de audio estéreo.
- Cuatro puertos USB 2.0 para conectar periféricos de todo tipo.
- Puerto serie RS232 para conectar dispositivos compatibles.
- Conector para tarjetas de memoria de tipo SD y MMC.

---

[53] http://elinux.org/Beagleboard:BeagleBoard
[54] http://en.wikipedia.org/wiki/File:BeagleBoard_xM.JPG

- Conector para cámaras.
- 28 pines de comunicaciones que incluyen conexiones de tipo: I2C, GPIO, PWM, MMC, UART, McBSP, MsSPI.

Los diseños del sistema también estarán disponibles para su descarga desde la wiki[55] del fabricante.

BeagleBone.

Con un diseño totalmente renovado la *BeagleBoard Foundation* en el año 2011 introdujo en el mercado la primera BeagleBone.

Concebida como una placa de expansión para las BeagleBoard, o como una placa de desarrollo por sí misma, esta placa dispone de una relación calidad-precio muy interesante que la hace apetecible para un gran número de proyectos.

Es interesante comentar que el diseño de esta placa permite usar placas de expansión pinchándolas en sus puertos de expansión. Un ejemplo similar de este caso lo representa Arduino con sus famosos *shields*.

La propia BeagleBoard Foundation recopila en su wiki un listado[56] de las placas de expansión más interesantes.

Figura 14. BeagleBone (BeagleBoard Foundation, 2013)[57].

Las características técnicas del dispositivo serán las siguientes:

- Texas Instrument SOC AM335x que cuenta con un microprocesador ARM® CORTEX™ corriendo a 720 megahercios.

[55] http://elinux.org/Beagleboard:BeagleBoard-xM
[56] http://elinux.org/Beagleboard:BeagleBone_Capes
[57] http://beagleboard.org/static/images/product_beaglebone.jpg

- 256 megabytes de memoria RAM DDR2.
- Alimentación mediante puerto microUSB OTG o a través de un conector de tipo cilíndrico.
- Interfaz Ethernet 10/100.
- Puerto USB 2.0 para conectar dispositivos de todo tipo.
- Conector para tarjetas de memoria microSD.
- Posibilidad de usar un puerto JTAG virtual mediante el puerto USB.
- Conector microHDMI en placa.
- Gran variedad de puertos de comunicaciones. Entre ellos podemos encontrar: 2 I2C, 5 UART, SPI, CAN, 66 GPIO, 8 PWM y 8 ADC.

BeagleBone Black.

Conservando el factor de forma del desarrollo anterior, la BeagleBone Black nace como una actualización de este último.

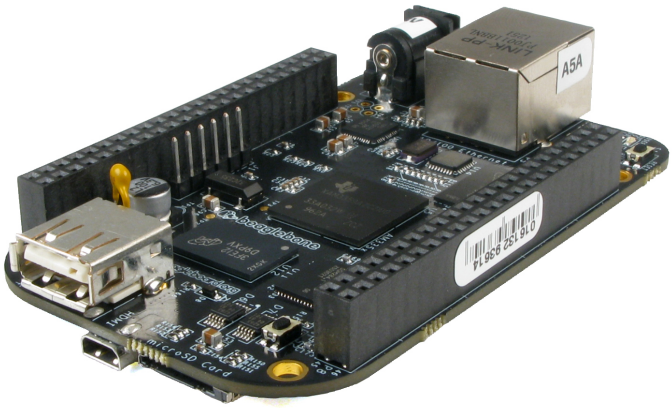


Figura 15. BeagleBone Black[58].

Debido a la forma similar la gran mayoría de los accesorios disponibles para la BeagleBone serán compatibles con la BeagleBone Black.

Como hemos comentado las características técnicas de este modelo serán superiores al anteriormente citado. Podemos definirlas en los siguientes puntos:

- Texas Instrument SOC AM335x con un microprocesador ARM® CORTEX™ corriendo a 1 gigahercio.
- 512 megabytes de memoria RAM de bajo consumo de tipo DDR3.
- 2 gigabytes de memoria eMMC integrada en placa.
- Alimentación disponible mediante conector de tipo cilíndrico o mediante un conector microUSB OTG.

[58] http://www.logicsupply.com/blog/2013/05/23/beaglebone/

- Interfaz Ethernet 10/100.
- Puerto USB 2.0 para conectar periféricos.
- Conector JTAG en placa.
- Posibilidad de ampliar la memoria del sistema mediante tarjetas microSD.
- Tres botones de acción para controlar diferentes opciones del sistema.
- Gran cantidad de puertos de comunicaciones del sistema. Entre ellos encontraremos puertos de tipo: McASP0, SPI1, I2C, GPIO, LCD, GPMC, MMC1, MMC2, 7 AIN, CAN0 y EHRPWM. Además posee cuatro timers y tres puertos serie.

Los primeros dos modelos, BeagleBoard y BeagleBoard-xM, serán los más caros. El precio de la BeagleBoard en España rondará los 185 euros costando la BeagleBoard-xM sobre 170 euros.
Las placas de la serie BeagleBone serán algo más económicas. El modelo original, Beaglebone, costará una media de 80 euros costando el siguiente modelo alrededor de 40 euros.
En los precios mencionados anteriormente no se tienen en cuenta los gastos de envío.

Olinuxino.

La última familia de dispositivos basados en microprocesadores ARM® que nombraremos son creados por la empresa Olimex Limited[59]. Esta empresa, localizada en Bulgaria, se dedica desde hace más de veinte años al diseño y fabricación de dispositivos de hardware libre para entornos industriales.
Estos años de desarrollo han llevado a Olimex Ltd. a poseer varias familias de modelos que podremos agrupar por el microprocesador del que disponen.

Para nuestro caso de estudio nos centraremos en los modelos agrupados por la denominación Olinuxino. Éstos, al igual que los dispositivos de los puntos anteriores, están preparados para utilizar una variedad de sistemas operativos basados en el kernel de Linux.
El sistema que puedan usar estará determinado por el hardware del dispositivo ya que, como podremos comprobar, la gran variedad de dispositivos fabricados por esta empresa conlleva una gran disparidad en sus especificaciones.

Un foro[60], abierto a todos los desarrolladores de sus modelos, y una wiki (Olimex Ltd., 2013)[61], donde detallan los pasos a seguir para la puesta a punto de sus dispositivos, conforman el soporte que ofrece esta compañía a sus clientes.
A continuación, agrupados por el microprocesador que portan, detallaremos las características técnicas de los dispositivos comercializados por esta compañía.

[59] https://www.olimex.com/
[60] https://www.olimex.com/forum/
[61] https://www.olimex.com/wiki/Main_Page

iMX233

Utilizado en un principio, este integrado permitió a *Olimex Limited* el diseño, y producción, de una primera placa de prototipado con capacidad de ejecutar un entorno de escritorio completo basado en Linux.

Los integrados de esta familia portan un microprocesador ARM9™ a una velocidad de 454 megahercios que puede redireccionar 64 megabytes de memoria RAM.
Estas características, claramente inferiores a las mencionadas anteriormente, se compensan con unos costes de producción realmente bajos. Inútiles para aplicaciones que requieran gran capacidad de cálculo son realmente útiles en entornos sencillos donde prime un presupuesto ajustado.

Olimex Ltd. ofrece en su página (Olimex Ltd., 2014)[62] web cinco productos englobados en esta categoría:

- iMX233-OLinuXino-MAXI.
- iMX233-OLinuXino-MICRO.
- iMX233-OLinuXino-MINI y iMX233-OLinuXino-MINI-WiFi.
- iMX233-OLinuXino-NANO.

En la siguiente tabla podemos encontrar un resumen de las características de los modelos mencionados anteriormente.

Modelo	MAXI	MICRO	MINI	MINI-WiFi	NANO
Microprocesador	ARM926J a 454 Mhz.				
RAM	64 megabytes.				
USB	2 del tipo 2.0.	1 del tipo 2.0.	3 del tipo 2.0.		1 del tipo 2.0.
Ethernet	10/100.	No.			
Conectividad inalámbrica	No.			WiFi RTL8188CU.	No.
Memoria disponible	Ranura para el almacenamiento de memoria fija mediante tarjetas microSD.				
Salida de vídeo	Salida de video en PAL/NTSC mediante conector RCA.				No.
Conexiones de audio	Entrada y salida de audio estéreo.	No.	Entrada y salida de audio estéreo.		No.
Puertos de expansión	40 puertos GPIO.	60 puertos GPIO.	40 puertos GPIO.		48 puertos GPIO.
Botones de control	2.	3.	2.		3.
Energía necesaria	[6, 16] VDC.	5 VDC.	[6, 16] VDC.		3.7 VDC.
Precio (antes de costes extra)	45 €.	24 €.	35 €.	45 €.	22 €.

Tabla 1. Caracterísiticas placas de desarrollo de la familia iMX233.

[62] https://www.olimex.com/Products/OLinuXino/iMX233/

En el CD-ROM que acompaña a esta memoria se encuentra el último manual disponible, a fecha de entrega de este proyecto, de cada placa de desarrollo mencionada en la anterior tabla.

A13

La utilización del SOC A13 supuso una gran mejora en la velocidad de procesamiento y la cantidad de memoria RAM que podían manejar los nuevos modelos.

Los modelos dentro de esta categoría son los siguientes:

- A13-OLinuXino-WIFI.
- A13-OLinuXino.
- A13-OLinuXino-MICRO.

Modelo	A13-OLinuXino-WiFi	A13-OLinuXino	A13-OLinuXino-MICRO
Microprocesador	ARM® CORTEX™ A8 a 1 Ghz. Chip gráfico Mali400.		
RAM	512 megabytes.		256 megabytes.
USB	3 USB hosts disponibles para el usuario.	3 USB hosts + conectores para un cuarto.	1 USB Host + 1 USB-OTG.
Ethernet	No.		
Conectividad inalámbrica	WiFi mediante RLT8188CU conectado a placa.	No.	
Memoria disponible	4 GB de memoria integrada. Conector microSD para ampliación.	Almacenamiento fijo mediante tarjetas microSD.	Almacenamiento fijo mediante tarjetas microSD.
Salida de vídeo	Salida mediante interfaz VGA disponible. Posibilidad de conectar una pantalla LCD deshabilitando la salida analógica.		
Conexiones de audio	Entrada de micrófono y salida de audio estéreo.		
Puertos de expansión	74 puertos disponibles para: añadir una memoria de tipo NAND, conectar una pantalla LCD, conectar tarjetas SD y para interfaces de tipo I2C, UART Y GPIO. También hay 5 pines para tareas de administrador.		
Botones de control	5.		1.
Energía necesaria	[6, 16] VDC. Posibilidad de alimentación mediante USB-OTG.	[6, 16] VDC.	5 VDC. Posibilidad de alimentación mediante USB-OTG.
Precio (antes de impuestos y gastos de envío)	55 €	45 €	35 €

Tabla 2. Caracterísiticas modelos A13 de Olimex ltd.

Tendremos también que mencionar que, tanto el modelo *A13-OLinuXino-WiFi* como el modelo inferior *A13-OLinuXino,* cuentan con un módulo RTC para proporcionar al sistema una hora estable frente a posibles cortes en la alimentación del dispositivo.
Tampoco podemos dejar pasar el hecho de que la mejora técnica de esta familia, con respecto a la anterior, posibilita la conexión directa de estas placas con pantallas LCD para mejorar la visualización de la información proporcionada por estos sistemas.

Los manuales de estos tres modelos también estarán disponibles en el CD-ROM que viene con esta memoria. Para más información habrá que acudir a la wiki mencionada en el primer epígrafe de este punto.

A10

El SOC A10 se puede considerar un modelo mejorado del SOC A13. Aunque ambos portan el mismo microprocesador el A10 posee un rango de conexiones mayor.

Dentro de esta categoría[63] solo existe un modelo disponible, el *A10-OLinuXino-LIME*.

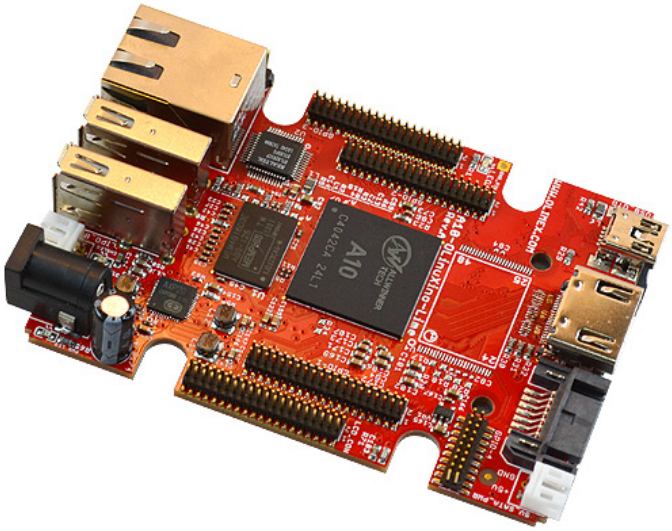


Figura 16. A10-OLinuXino-LIME (Olimex Ltd., 2014)[64].

Contando con un ARM® CORTEX™ A8 a 1 gigahercio como microprocesador y con un chip gráfico Mali 400 las características técnicas de este dispositivo serán las siguientes:

- 512 megabytes de memoria RAM de tipo DDR3.
- Interfaz SATA con alimentación incorporada para discos duros de 2’5 pulgadas.
- Conector HDMI para la salida de vídeo.
- Dos puertos USB 2.0 protegidos contra las sobrecargas.
- Un puerto USB-OTG con capacidad de alimentar al dispositivo.

[63] https://www.olimex.com/Products/OLinuXino/A10/
[64] https://www.olimex.com/Products/OLinuXino/A10/A10-OLinuXino-LIME/open-source-hardware

- Interfaz Ethernet 10/100.
- Conector microSD para ampliar la memoria del dispositivo.
- Interfaz LiPo para la alimentación mediante baterías externas.
- Alimentación también disponible mediante puerto USB-OTG o mediante un conector de tipo cilíndrico.
- 160 puertos de tipo GPIO en cuatro conectores distintos.
- 4 botones de control.
- 3 leds para visualizar diferentes eventos.

Con un precio de venta unitario, sin costes de envío e impuestos, de 30 € por unidad este producto es una opción muy interesante por su excelente relación calidad-precio.

A10S

Aunque aún se encuentren listados en la web (Olimex Ltd., 2014)[65] de la compañía los modelos de esta categoría se consideran obsoletos. Esto es debido a que la empresa fabricante del integrado ha decidido dejar de prestar soporte a éste.

Los modelos que esta compañía ofrecía eran:

- A10S-OLinuXino-MICRO.
- A10S-OLinuXino-MICRO-4GB.

Desde este mismo sitio recomiendan dejar apartados todos los proyectos que involucren a estos modelos y no empezar ningún desarrollo nuevo con ellos.

A20

Partiendo del diseño de las placas anteriores se realizó una mejora que, fundamentalmente, implicaba un cambio de SOC para lograr un mayor rendimiento.

El SOC A20 al contrario que el SOC anterior cuenta con un microprocesador de doble núcleo. La cantidad de memoria RAM gestionada por el integrado también es considerablemente mayor.

La disposición de los pines y el tamaño de las placas es idéntico a los dos modelos anteriores. Esto hace que la mayoría de los complementos disponibles para el grupo anterior estén disponibles también para este.
Actualmente se encuentra en desarrollo una nueva placa con un SOC A20 utilizando el layout de la *A10-OLinuXino-LIME.*

[65] https://www.olimex.com/Products/OLinuXino/A10S/

Los modelos disponibles actualmente son los siguientes:

- A20-OLinuXino-MICRO.
- A20-OLinuXino-MICRO-4GB.

La única diferencia existente entre ambos modelos radica en que el segundo modelo dispone de un módulo de 4 gigabytes de memoria integrado en la misma placa.

Así, acorde a su web (Olimex Ltd., 2014)[66], las características comunes de ambos desarrollos se pueden resumir en los siguientes puntos.

- SOC A20 que integra un microprocesador ARM® CORTEX™ A7 de doble núcleo con un chip gráfico de doble núcleo Mali400.
- Un gigabyte de memoria RAM de tipo DDR3.
- Conector SATA con alimentación de corriente incorporada para discos duros de 2'5 pulgadas.
- Dos conectores para utilizar hasta dos tarjetas de memoria para almacenamiento permanente. Una de ellas tendrá que ser de tipo microSD siendo la otra de tipo SD.
- 2 puertos USB de alta velocidad con protección para sobretensiones.
- Salida de video mediante interfaz HDMI o a través de VGA.
- Interfaz Ethernet 10/100.
- Salida de audio estéreo para auriculares.
- Entrada de línea para audio estéreo.
- Tres leds para mostrar información sobre los puertos GPIO, el estado de carga de la batería y el suministro de alimentación del sistema.
- 160 pines destinados a puertos GPIO dispuestos en tres conectores.
- Conectores para depuración del sistema en placa.
- Dos conectores UEXT para módulos de expansión.
- 11 botones para tareas de administración del sistema.
- Compatibilidad con pantallas LCD comercializadas por la misma empresa fabricante de esta placa.
- Posibilidad de alimentación mediante conector de tipo cilíndrico (tensión de entrada entre 6 y 16 voltios de corriente continua), mediante puerto USB-OTG y mediante batería externa a través de una interfaz LiPo.

[66] https://www.olimex.com/Products/OLinuXino/A20/

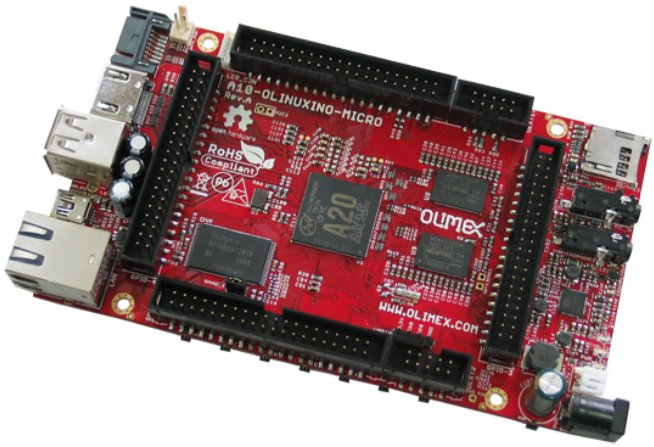


Figura 17. A20-OLinuXino-MICRO (Olimex Ltd., 2014) [67].

Con un precio de 55 euros para el modelo más sencillo y costando 65 euros el modelo con la memoria integrada estos dos modelos se postulan como los más caros de esta compañía. A estos precios tendremos que añadirle impuestos y costes de envío.

Galileo.

La placa de desarrollo Galileo se puede considerar la rara avis de este pequeño muestrario de dispositivos.

Con esta placa Intel y Arduino buscaban crear un producto compatible pin a pin con las placas de desarrollo creadas para la plataforma Arduino. Intel intenta así aprovechar el tirón que tienen este tipo de dispositivos en ambientes de experimentación y enseñanza.

Galileo posee los mismos pines dispuestos de la misma forma que la placa Arduino Uno. Esto quiere decir que los shields disponibles para la placa fabricada por Arduino estarán disponibles para Galileo.

[67] https://www.olimex.com/Products/OLinuXino/A20/A20-OLinuXino-MICRO/open-source-hardware

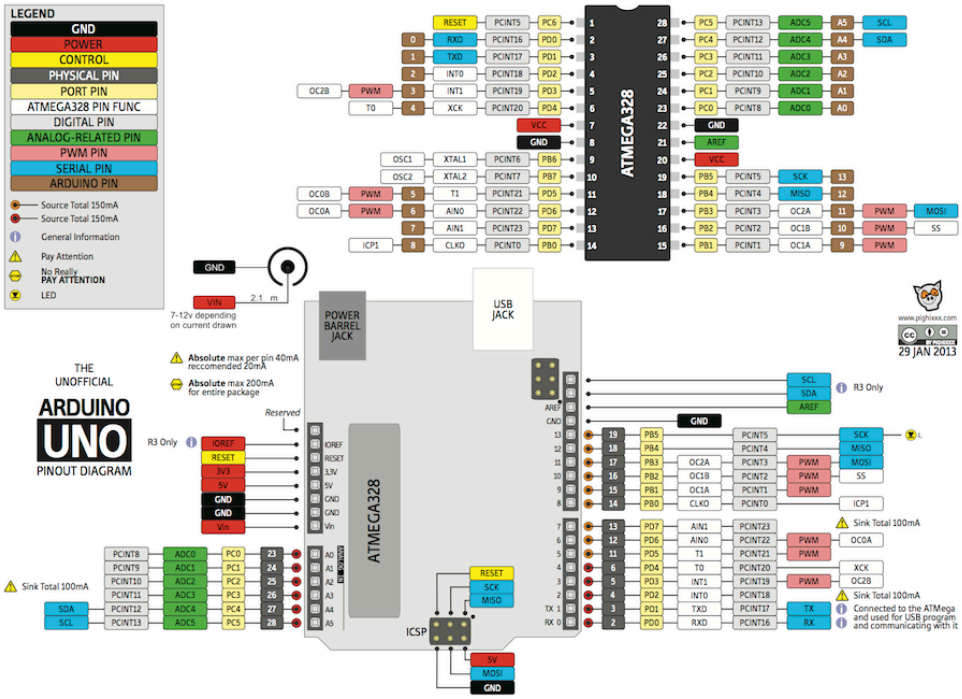


Figura 18. Diagrama de pines Arduino Uno (pighixxx, 2013) [68].

Aunque no oficial, el diagrama anterior nos da una clara imagen de la variedad de conexiones disponibles en este sistema.

La placa de desarrollo Galileo nos proporciona las mismas conexiones dándonos, gracias a su SOC Intel®, una capacidad de cálculo mucho mayor.

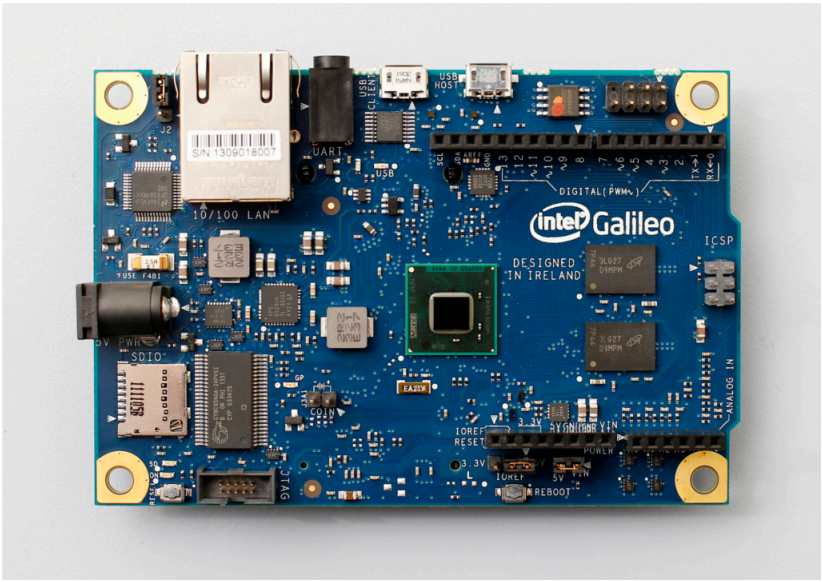


Figura 19. Intel Galileo (Arduino, 2014) [69].

[68] http://forum.arduino.cc/index.php/topic,146315.0.html

Las características técnicas de Galileo serán, acorde a su página web, las siguientes:

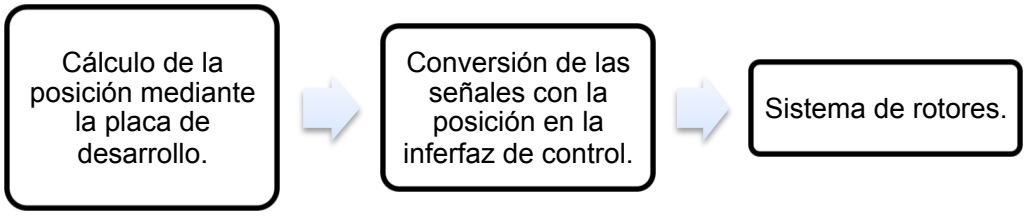
- SOC Intel® Quark SOC X1000 que cuenta con un microprocesador Pentium 32-bit corriendo a 400 megahercios.
- Interfaz Ethernet 10/100.
- Un puerto USB 2.0 del tipo Host utilizado para conectar dispositivos.
- Un puerto USB 2.0 del tipo Client utilizado para la programación del dispositivo.
- Puerto miniPCIe para la conexión de dispositivos externos.
- Conector para el uso de tarjetas microSD para el almacenamiento masivo de memoria.
- Puerto JTAG para la depuración del sistema.
- Alimentación mediante conector de tipo cilíndrico. La tensión requerida será de cinco voltios, siendo la corriente mínima necesaria de tres amperios.

A estas características añadiremos las conexiones mencionadas anteriormente.
También tendremos que comentar que Galileo, al igual que los sistemas anteriores, soporta una amplia variedad de sistemas basados en el kernel de Linux.

El precio unitario de esta placa, antes de impuestos y costes de envío será de, aproximadamente, 55 euros.
Si necesitamos más información siempre podremos acudir a los foros[70] oficiales de Arduino o al FAQ[71] desarrollado por Intel®. El datasheet desarrollado por Intel® estará también disponible en la documentación de esta memoria.

Interfaces de conexión actuales.

Nuestra placa de desarrollo no se podrá conectar directamente al sistema de rotores. Para ello necesitará una interfaz hardware que acondicione las señales de control al sistema de potencia de los motores.



[69] http://arduino.cc/en/ArduinoCertified/IntelGalileo
[70] http://forum.arduino.cc/
[71] https://communities.intel.com/thread/45225/45225

Para este cometido tendremos varios desarrollos disponibles. Cada uno de estos desarrollos estará preparado para un sistema de rotores diferente que contará con su protocolo correspondiente.

Al ser este un campo no tan popular como los anteriores, las opciones open source disponibles serán mucho más escasas. Aun así el abanico es lo suficientemente amplio como para tener una base para nuestro prototipo.
A continuación pasaremos a detallar las características técnicas de los modelos más interesantes. En el siguiente capítulo decidiremos que opción es la mejor para nuestro desarrollo.

FODTrack

El primer desarrollo que mencionaremos, y posiblemente el más antiguo, será *FodTrack*. Aunque creado como un proyecto personal en el año 1994, no vería la luz pública hasta unos años después cuando el desarrollador decidió publicar la primera versión estable del conjunto.

Este desarrollo comprende el software de guiado y la interfaz hardware de conexión al sistema de rotores.
Desechamos la idea de exponer el software en el punto anterior debido a la naturaleza cerrada del programa de control. El diseño del hardware, sin embargo, está disponible en la página[72] web personal del desarrollador. Como no posee sin licencia de ninguno tipo, el creador del mismo lo deja a disposición del que desee usarlo.

[72] http://ludens.cl/Electron/fodtrack/fodtrack.html

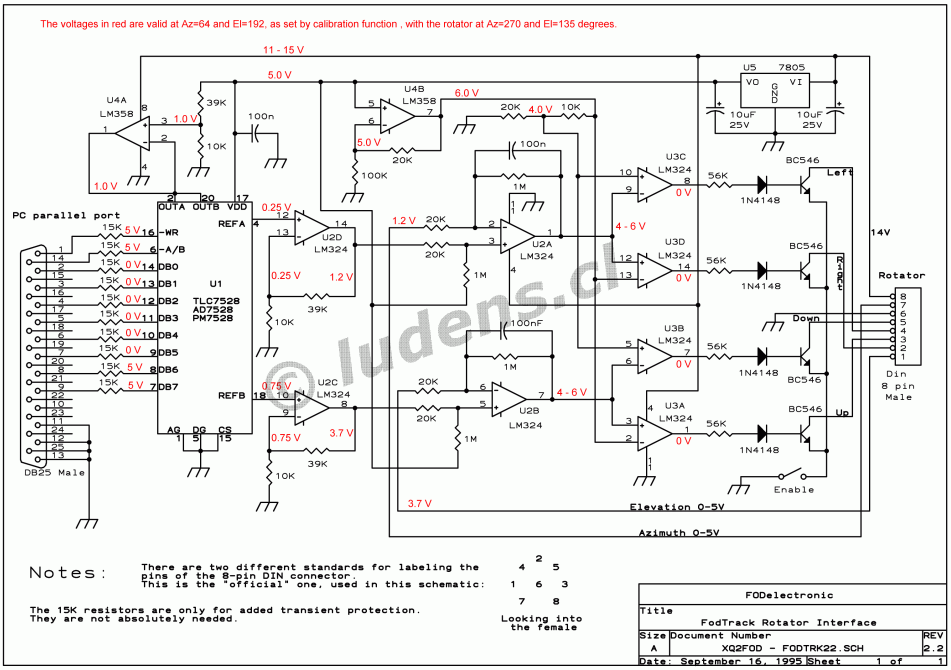


Figura 20. Circuito de control del sistema FODTrack (XQ2FOD, 1995)[73].

Acorde al manual del producto, este sistema será compatible con los modelos de Yaesu 5400, 5500 y 5600. Para ello solo tendremos que conectar la salida del circuito a la entrada de control externo de estos sistemas.
Este sistema también ofrece la posibilidad de controlar otros sistemas reajustando la configuración del mismo. Para ello habrá que modificar un archivo de texto del sistema.

Debido a la antigüedad de este desarrollo la interfaz de conexión del ordenador a la placa de control se hará mediante un puerto paralelo DB25. Analizaremos los posibles inconvenientes en el siguiente capítulo.
El software de FODTrack también estará disponible en el CD-ROM que acompaña a esta memoria.

Las Vegas Boulevard Tracker.

Presentado el año 2003 en el coloquio de la sección británica de AMSAT pretendía solucionar algunas limitaciones de los controladores existentes en el mercado.

Al ser un proyecto de naturaleza open source el diseño del circuito y el código fuente para la programación del microcontrolador están disponibles en la página (G6LVB, LVB Tracker, 2004)[74] web del desarrollador.

[73] http://ludens.cl/Electron/fodtrack/fodtrack.html
[74] http://www.g6lvb.com/articles/lvbtracker/v010.zip

El creador de este sistema ofrece también la posibilidad de comprar placas con los componentes ya ensamblados desde la tienda[75] virtual de AMSAT.

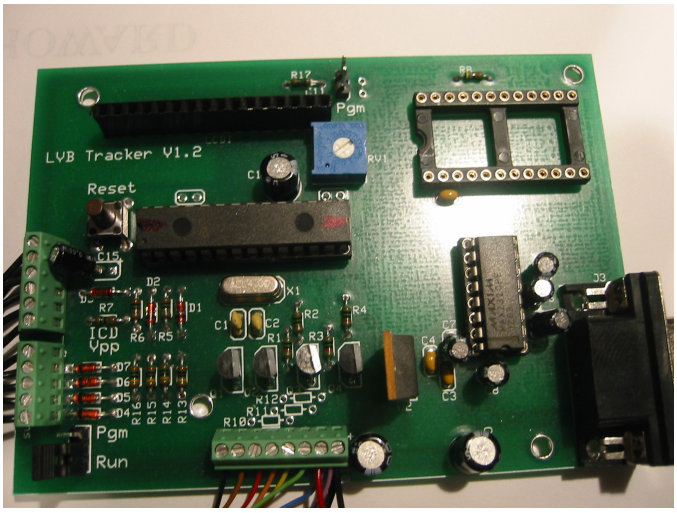


Figura 21. LVB Tracker V1.2[76] (G6LVB H. , 2004)

Acorde a la información proporcionada por el creador, podremos definir este proyecto en los siguientes puntos:

- Barato. El coste de un controlador comercial (Yaesu GS-232) ronda los 700 euros. El precio de los componentes de esta placa ronda los 12 euros.
- Distintas opciones de conexión. Aunque en la foto superior esté usando para su comunicación un puerto serie RS-232, esta placa está preparada para usar como interfaz de conexión un puerto USB o, incluso, un puerto Ethernet.
- Los protocolos soportados por este dispositivo serán los dos más típicos en el mercado: *GS-232* y *Easycomm*. Eso nos permitirá usarlo para una amplia variedad de rotores.
- El microcontrolador se puede programar desde la misma placa de control. Eso evita tener que adquirir más dispositivos.
- Compatibilidad con diferentes sistemas operativos. El hecho de usar un protocolo estándar nos permite utilizar esta placa con software de sistemas operativos muy diversos.
- El microcontrolador también está preparado para mostrar información relativa al sistema mediante un display LCD.

Como es habitual, en el siguiente capítulo compararemos este producto con los demás existentes. Los esquemáticos del circuito, el layout de la placa y los documentos informativos aportados por el creador están disponibles en el CD-ROM de la memoria.

[75] http://shop.amsat.org.uk/
[76] http://www.g6lvb.com/articles/lvbtracker/

Las Vegas Boulevard Tracker 2.

Nacido como una actualización del anterior proyecto, *Las Vegas Boulevard Tracker 2*, en adelante *LVB Tracker 2*, es la respuesta de muchos usuarios a las limitaciones del anterior sistema.

Figura 22. LVB Tracker 2. (G6LVB, LVB Tracker, 2005)[77]

Utilizando como base la placa del anterior desarrollo esta nueva versión cambia el microcontrolador para así disponer de más funcionalidades

Acudiremos a la página[77] web del creador para conocer las nuevas funcionalidades que aporta este prototipo:

- Seguimiento autónomo. A diferencia del sistema anterior este modelo no necesita un ordenador para funcionar. Esta funcionalidad se ha podido conseguir gracias al desarrollo de nuevos algoritmos de cálculo.
- Compatibilidad con dispositivos GPS. Esto permitirá al sistema conocer la localización y hora local del usuario.
- El sistema, reconociendo la posición del observador y calculando la posición de los objetos, nos avisará de qué satélites son visibles en ese instante. Mediante un menú de selección podremos elegir que objeto queremos que nuestro sistema siga.
- Múltiples datos disponibles sobre el satélite seleccionado. Estas incluyen:
    - Acimut y altura actual
    - Hora y fecha del próximo orto, *AOS*, del objeto seleccionado. Tiempo restante hasta éste.
    - Hora y tiempo restante hasta el próximo ocaso, *LOS*, del satélite elegido.
    - Acimut del satélite en el instante del orto y del ocaso.
    - Duración de la siguiente pasada del satélite sobre el observador.
    - Rango de valores de altura sobre la superficie terrestre del objeto. Altura actual sobre la superficie de la tierra.
    - Número de órbita e inclinación de ésta.

[77] http://www.g6lvb.com/articles/LVBTracker2/index.htm

- Posibilidad de actualizar los elementos orbitales mediante un cliente terminal o telnet.
- Alertas sonoras para diversos eventos.
- Posibilidad de funcionar mediante baterías. Dos baterías del tipo AAA nos permitirán un uso continuado de alrededor de nueve horas.

Los protocolos de conexión disponibles serán los mismos que antes, esto es *GS-232* y *Easycomm*. Estos nos permitirán una compatibilidad total con antiguos dispositivos.

Aunque actualmente en desarrollo, desde la página web del creador tenemos a nuestra disposición el código fuente para programar el pic. También tenemos disponible para su descarga información sobre el sistema.

El precio de este sistema será muy similar al del anterior ejemplo. Para cualquier duda podremos acudir al creador del proyecto.

WRAPS.

Creado a finales del año 2013, *WRAPS* consiste en una alternativa a sistemas de motores como el Yaesu G-5500.
La idea del creador del proyecto era ofrecer a los estudiantes un sistema de rotores mucho más barato que les permitiera iniciarse en el mundo de la radioafición amateur con un desembolso considerablemente menor. Acorde a la información proporcionada por éste, el coste de las piezas necesarias para la construcción del sistema ronda los 275 dólares. El precio de un sistema profesional como el mencionado anteriormente no baja de los 1300 dólares.

Para facilitar la utilización de este sistema, los sistemas de software compatibles con este dispositivo son los más usuales. Así podemos encontrar el protocolo EasyComm (Chris Jackson)[78], SatPc32[79] o SAEBreTrack[80].
El hecho de poder alimentar al sistema con una fuente de corriente continua de doce voltios, incluso con una pila de nueve voltios, incrementa la facilidad de uso del sistema.

No todo podían ser ventajas, y es que el bajo coste del sistema imposibilita a este para algunas funciones. Podemos resumir sus flaquezas en los tres puntos siguientes:

- Las antenas tendrán que ser muy ligeras. La construcción mecánica del sistema impide usar conjuntos pesados de antenas.
- Imposibilidad para funcionar de forma continua. La calidad de los materiales utilizados no es suficiente para soportar un uso continuado.
- Falta de impermeabilización. El sistema no podrá ser utilizado a la intemperie.

[78] ftp://www.amsat.org/amsat/software/win32/wisp/easycomm.txt
[79] http://www.dk1tb.de/indexeng.htm
[80] https://sites.google.com/site/marklhammond/saebrtrack

El siguiente esquema nos puede dar una idea más clara del funcionamiento del dispositivo.

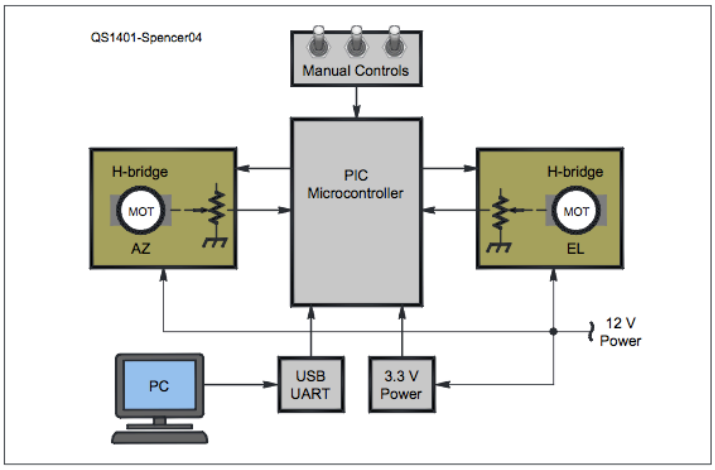


Figura 23. Diagrama de funcionamiento de WRAPS[81].

Podremos ampliar la información sobre este desarrollo en el artículo (Mark Spencer, 2014)[82] de referencia publicado en la web de la ARRL.
Los esquemas de fabricación están disponibles para su uso libre en entornos educativos y no corporativos.

[81] http://ww2.amsat.org/xtra/wraps-mark-spencer-wa8sme-qst-jan-2014-copyright-arrl.pdf
[82] http://ww2.amsat.org/xtra/wraps-mark-spencer-wa8sme-qst-jan-2014-copyright-arrl.pdf

3 - Objetivos de este proyecto.

Objetivos de este proyecto.

Cada parte del proyecto tendrá que cumplir una serie de funcionalidades. Cuando hablamos de funcionalidades nos estamos refiriendo tanto a las capacidades del hardware, como a las habilidades del software para calcular posiciones o emitir órdenes de movimiento.

En los siguientes puntos esbozaremos las funcionalidades requeridas para cada parte del sistema. Así podremos seleccionar el hardware, el software y los servicios necesarios para llevar a cabo nuestro desarrollo.

Objetivo final.

El objetivo final de este proyecto es el desarrollo de un software de control que, a través de una interfaz gráfica sencilla, nos permita controlar un sistema de rotores. Este objetivo puede verse de una manera más clara plasmado en el siguiente esquema.

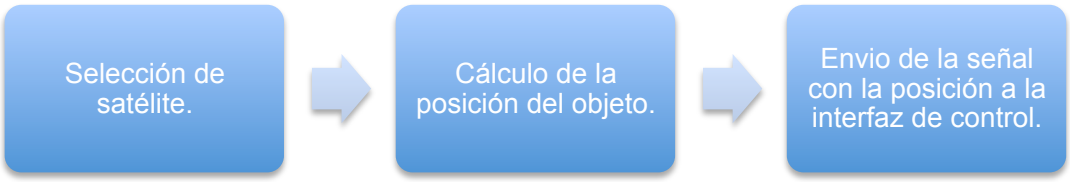


Figura 24. Esquema de funcionamiento del sistema.

¿Qué necesitaremos para cumplir este objetivo?

En primer lugar tendremos que recalcar que nuestro desarrollo será multiplataforma. Aunque en esta memoria usaremos como dispositivo de pruebas una placa con un sistema operativo de base Linux, nuestro objetivo es que cualquiera pueda usar este programa en su ordenador, sea del tipo que sea.
Como parte del proyecto comprobaremos que el código es correcto usándolo en los tres sistemas operativos principales: Windows, Mac OS X y Linux.

La idea de usar un soporte hardware libre es disponer de un medio rápido y cómodo para desarrollar esta nueva idea.

Para ser consecuentes con la filosofía open-source de nuestro código buscaremos un dispositivo con el mayor número de componentes *libres*.

Además este sistema tendrá que tener un mínimo de potencia para que un sistema operativo completo funcione sin problemas. Esto nos dará la posibilidad de ampliar nuestro programa o utilizar a la misma vez algún tipo de software de control sin problema alguno.

Como el lenguaje de programación que usaremos será Python, también tendremos que cerciorarnos de que la placa de desarrollo posee las librerías necesarias para utilizarse bajo ese lenguaje. Puertos de comunicaciones, librerías de controladores, software de testeo y ayuda al desarrollador deberán estar disponibles para el sistema elegido.

Una vez creado el software tendremos que realizar las pruebas de control del mismo. Para realizarlas la placa tendrá que ser capaz de comunicarse con el exterior.
Los puertos de comunicación más usuales en las interfaces de control de rotores son RS232 y USB. Nuestro dispositivo tendrá que poseer alguno de ellos para sernos de utilidad.

¿Cuándo sabremos que hemos alcanzado nuestro objetivo?

Como último paso tendremos que testear el funcionamiento de todo el conjunto. Para eso conectaremos nuestro dispositivo a un sistema de rotores real o, en su defecto, realizaremos simulaciones con un osciloscopio.

Cuando el movimiento de los motores, o la salida del osciloscopio, se correspondan con lo calculado teóricamente daremos por finalizado el desarrollo del proyecto.
Aunque después de esto no realizaremos más cambios a nuestro código o a nuestro hardware, plantearemos una serie posibles mejoras en el último capítulo de esta memoria.

Conclusión

En el capítulo anterior realizamos un breve repaso por las opciones más interesantes para el desarrollo de nuestro proyecto.
Nos tocará ahora, conociendo los objetivos marcados, ver qué dispositivos y qué programas se ajustan más a ellos.

Dicho esto comenzamos con la elección de nuestros componentes.

Elección del servidor de datos orbitales.

No menos importante será la decisión de que servidor de datos orbitales usar.
A continuación analizaremos las dos opciones mencionadas en el anterior capítulo sopesando los pros y los contras de cada una de ellas.

Celestrak.

La opción más veterana de ambas es quizás la más popular, la facilidad de acceso a su repositorio de datos le ha granjeado la simpatía de muchos desarrolladores de software.
Esto es debido, principalmente, a que *Celestrak* no requiere de registro para su utilización.
Para facilitar la tarea a los usuarios, los documentos de texto que contienen los datos orbitales se encuentran en direcciones fijas. Para acceder a ellos de manera periódica bastará con guardar la dirección del archivo deseado.

Pero la sencillez de este proyecto es también su mayor tara. Y es que *Celestrak* no posee una API propia. El acceso a los elementos orbitales de un objeto o cualquier otra consulta específica tendrá que ser desarrollada por el programador.

Tampoco cuenta con una empresa detrás y es que este proyecto es mantenido por un solo individuo.

Spacetrack.

Siendo una opción mucho más reciente, *Spacetrack* pretende corregir algunas de las deficiencias de *Celestrak* y proporcionar al usuario una experiencia más completa.

Al contrario que el anterior servicio, este proyecto posee una API propia para desarrolladores. Esta nos permitirá acceder a un dato en particular o crear listas con los objetos de nuestro interés.
El punto negativo de esta API es que requerirá del registro del usuario para su uso. Este registro tendrá que ser aprobado por *Spacetrack*.

Los datos orbitales que nos proporcionarán ambos servicios serán los mismos ya que estos proveerán del mismo organismo.

Conclusión.

Nuestra opción final será *Celestrak*. Los dos motivos en los que se fundamenta este razonamiento son los siguientes:

- No requiere registro. Esto posibilita que cualquier usuario utilice nuestro software sin tener que dar datos personales.

- La API de Spacetrack es innecesaria en nuestro proyecto. Nuestro software se encargará de extraer los datos necesarios de los ficheros de elementos orbitales.

Elección de las placas de desarrollo.

Para facilitar la elección de una placa de prototipado para nuestro desarrollo nos centraremos en los tres puntos principales de este.
Primero trataremos de resumir de la manera más clara posible el hardware disponible en cada placa mencionada anteriormente. Pondremos nuestra atención en los elementos más necesarios.
A continuación daremos una visión de la comunidad de desarrolladores presente detrás de cada dispositivo. Esto nos dará una idea del "soporte" del producto.
Por último lugar, y no menos importante, listaremos las licencias bajo las cuales se encuentran cada uno de los sistemas. Escoger un producto con una licencia adecuada nos puede ahorrar más de un problema en el futuro.

Comparativa de características.

La mejor manera de ver las capacidades de cada dispositivo será presentar estas en una tabla. Analizaremos los parámetros necesarios para nuestro proyecto, estos son: microprocesador, memoria RAM, almacenamiento externo, conectividad de red y disponibilidad de puertos USB, Ethernet, RS-232 y UART-TTL

Modelo	Microprocesador	RAM	Almacenamiento	Conectividad	Precio
Raspberry Pi – Model A	Broadcomm BCM2835.	256 MB.	Tarjeta SD.	USB, UART-TTL, Ethernet.	30 €.
Raspberry Pi – Model B	Broadcomm BCM2835.	512 MB.	Tarjeta SD.	USB, UART-TTL, Ethernet.	39 €.
CubieBoard	SOC A10, ARM® CORTEX™ A8 de un núcleo.	1 GB.	Tarjeta SD, SATA, 4 GB en placa.	USB, UART-TTL, Ethernet.	65 €.
CubieBoard2	SOC A20, ARM® CORTEX™ A7 de doble núcleo.	1 GB.	Tarjeta SD, SATA, 4 GB en placa.	USB, UART-TTL, Ethernet.	80 €.
CubieTruck (Cubieboard3)	SOC A20, ARM® CORTEX™ A7 de doble núcleo.	1 o 2 GB.	Tarjeta SD, SATA, 8 GB en placa.	USB, Ethernet, UART-TTL.	120 €.
BeagleBoard	SOC OMAP3530, ARM®CORTEX™ 8 de un núcleo.	256 MB.	Tarjeta SD, 512 MB en placa.	USB, UART-TTL, Ethernet, RS-232.	185 €.
BeagleBoard-xM	SOC AM37x, ARM® CORTEX™ A8 de un núcleo.	512 MB.	Tarjeta SD.	USB, UART-TTL, Ethernet, RS-232.	170 €.
BeagleBone	SOC AM335x ARM® CORTEX™.	256 MB.	Tarjeta microSD.	USB, UART-TTL, Ethernet.	80 €.

BeagleBone Black	SOC AM335x ARM® CORTEX™.	512 MB.	Tarjeta microSD, 2 GB en placa.	USB, UART-TTL, Ethernet.	40 €.
iMX233-MAXI	ARM926J.	64 MB.	Tarjeta microSD.	USB, UART-TTL, Ethernet.	45 €.
iMX233-MICRO	ARM926J.	64 MB.	Tarjeta microSD.	USB, UART-TTL.	24 €.
iMX233-MICRO-WIFI	ARM926J.	64 MB.	Tarjeta microSD.	USB, Wii, UART-TTL.	35 €.
iMX233-MINI	ARM926J.	64 MB.	Tarjeta microSD.	USB, UART-TTL.	45 €.
iMX233-NANO	ARM926J.	64 MB.	Tarjeta microSD.	USB, UART-TTL.	22 €.
A13-OLinuXino-WIFI	SOC A13 ARM® CORTEX™ A8 a 1 Ghz.	512 MB.	Tarjeta microSD, 4 GB en placa.	USB, UART-TTL, WiFi	55 €.
A13-OLinuXino	SOC A13 ARM® CORTEX™ A8.	512 MB.	Tarjeta microSD.	USB, UART-TTL.	45 €.
A13-OLinuXino-MICRO	SOC A13 ARM® CORTEX™ A8.	256 MB.	Tarjeta microSD.	USB, UART-TTL.	35 €.
A10-OLinuXino-LIME	SOC A10 ARM® CORTEX™ A8.	512 MB DDR3.	Tarjeta microSD, SATA.	USB, UART–TTL, Ethernet.	30 €.
A20-OLinuXino-MICRO	SOC A20 ARM® CORTEX™ A7 de doble núcleo.	1 GB.	Tarjeta SD y/o microSD, SATA.	USB, UART-TTL, Ethernet.	55 €.
A20-OLinuXino-MICRO-4GB	SOC A20 ARM® CORTEX™ A7 de doble núcleo.	1 GB.	Tarjeta SD y/o microSD, SATA, 4 GB en placa.	USB, UART-TTL, Ethernet.	65 €.
Galileo	Intel® Quark SOC X1000	256 MB.	Tarjeta microSD.	USB, UART-TTL, Ethernet.	55 €.

Tabla 3. Características de las placas de desarrollo.

Aunque no disponible en las interfaces de control mencionadas anteriormente podremos comprobar que hemos incluido el puerto UART-TTL en nuestra comparativa.
Esto es debido a lo fácil, y económico, que resulta realizar un convertidor de niveles TTL a niveles RS232. Para ello solo necesitaremos un integrado MAX3232 y unos cuantos elementos acondicionadores de la tensión.

Atendiendo a un criterio económico podemos comprobar como las opciones más interesantes serán: *Raspberry Model B*, *BeagleBone Black* y *A10-OLinuXino-LIME*.

Descartamos todos los productos pertenecientes a la familia iMX233 por ser componentes antiguos y con un rendimiento pobre para correr un escritorio completo.

Las tres opciones comentadas disponen de la conectividad necesaria para nuestro desarrollo, además de un potente microprocesador. Estos microprocesadores están apoyados por una cantidad más que suficiente de memoria RAM de última generación.

Comparativa de comunidad de usuarios.

Una comunidad de usuarios grande facilitará la creación de un prototipo. Sucederá con frecuencia que los problemas que nos surjan ya hayan sido planteados por algún otro usuario del producto.

A continuación, punto a punto, listaremos cada una de los productos mencionados anteriormente. Intentaremos dar una idea del soporte ofrecido por la compañía fabricante y por la comunidad amateur de desarrolladores.

Raspberry Pi.

Una de las maneras que tiene la Raspberry Pi Foundation para fomentar el desarrollo de aplicaciones para su dispositivo es a través de un foro[83] hospedado en su página oficial.
Con más de medio millón de mensajes, en el momento de escribir estas líneas, se puede considerar un referente para la consulta de dudas acerca de este sistema.

Como complemento a este foro existe una wiki[84] creada y mantenida por los miembros del anterior foro. En esta wiki se puede consultar desde procedimientos para instalar sistemas operativos en el dispositivo hasta los accesorios más populares de este.

La actividad de la comunidad de usuarios puede verse reflejada en la gran cantidad de blogs y páginas nacidas a raíz de este proyecto. También ha motivado la creación de tiendas on-line donde se venden toda clase de complementos para esta pequeña placa de desarrollo.

CubieBoard.

Cubietech Limited también intenta desde su página oficial ofrecer soporte a los usuarios de sus productos.
Desde esta página podremos acceder al foro[85] oficial donde, actualmente, hay más de doce mil mensajes referidos a estos dispositivos. Complementando al foro existe una lista de correo, un canal de chat de IRC, una wiki desarrollada de manera comunitaria y una

[83] http://www.raspberrypi.org/forum/
[84] http://elinux.org/RaspberryPiBoard
[85] http://www.cubieforums.com/

página de Google+. Podremos acceder a todos estos servicios desde la sección[86] de soporte de su sitio web.

Pese a todos los esfuerzos de esta compañía la comunidad de usuarios es notablemente inferior a la del producto anterior. Aun así no será difícil dar con tiendas on-line que suministren accesorios para este dispositivo o con blogs de aficionados donde muestren sus proyectos.

BeagleBoard.

La BeagleBoard Foundation, nacida como una fundación sin ánimo de lucro, ha intentado siempre facilitar la labor de sus desarrolladores, en su mayor parte estudiantes.

A las páginas dedicadas al soporte del hardware y del software creadas por la fundación tendremos que añadir foros, blogs y chats que, aunque hospedados en el sitio web oficial, son alimentados de contenidos por los desarrolladores de estos productos.
En las fichas de los dispositivos también podremos encontrar libros relacionados sobre cada producto y una pequeña relación de los proyectos más interesantes que se pueden realizar con estos.

Un listado general con todos los desarrollos existentes se puede encontrar en el mismo sitio[87] web. Desde aquí podemos contactar con otros programadores para unirnos a sus proyectos.

Todas estas políticas han conseguido que la comunidad de programadores que usan estos dispositivos tenga un tamaño respetable.

OLinuXino.

Como soporte al programador Olimex Ltd. mantiene un foro y una wiki en su sitio web.
Aunque la cantidad de usuarios del foro es considerablemente inferior a los casos anteriores Olimex Ltd. subsana el problema ofreciendo una wiki[88] con una gran cantidad de información.

En esta wiki podremos encontrar desde las características del hardware de cada uno de los dispositivos hasta métodos para la instalación de sistemas operativos en estos.

Olimex Ltd. también posee, como en el ejemplo anterior, un directorio actualizado de proyectos basados en sus productos. Además, para motivar a sus clientes, publica retos en la web de la empresa con relativa frecuencia.

---

[86] http://cubieboard.org/support/
[87] http://beagleboard.org/project
[88] https://www.olimex.com/wiki/

Como curiosidad tendremos que citar que también mantienen un archivo[89] de las conversaciones acontecidas en su canal de IRC.

Galileo.

Al ser un proyecto de muy reciente creación la comunidad de desarrolladores es aún muy limitada. Es de prever en que en los próximos meses el número de páginas relacionadas con este producto aumente.
En la actualidad pese a ser un producto certificado para Arduino no dispone siquiera de una sección propia en el foro de este. La mayor ayuda por parte de otros usuarios la podremos encontrar en los foros[90] de desarrolladores de Intel.

Para intentar suplir esta falta de contenido en la red la misma compañía ha colocado en la red algunos cursos relacionados con este hardware. Estos, libres y gratuitos, son accesibles desde la página (Intel Corporation ©)[91] de Intel.

Conclusión.

Aunque, como podemos comprobar, las empresas fabricantes han hecho un gran esfuerzo por ofrecer un soporte a sus comunidades de desarrolladores la placa de prototipado Raspberry Pi es la opción más aconsejable en este punto.

Las más de dos millones de placas de desarrollo vendidas han conseguido que la cantidad de blogs, foros, podcasts y videos online existentes sea mucho mayor que los disponibles sobre los productos de la competencia.

Comparativa de licencias.

Explicaremos, familia a familia de dispositivos, bajo que licencia se encuentran y que usos nos permite cada uno de ellas. Una licencia inadecuada para nuestro desarrollo podría hacernos caer en una trampa legal. Podría prohibirnos la futura distribución del prototipo que realicemos.

Raspberry Pi.

Las restricciones que impone la Raspberry Pi Foundation las podremos encontrar en un artículo (liz, Starting a business with a Raspberry Pi | Raspberry Pi, 2012)[92] de su blog oficial. Acorde al texto, el único requisito para comercializar un

[89] https://www.olimex.com/irc
[90] https://communities.intel.com/community/makers
[91] http://intel-software-academic-program.com/courses/#diy
[92] http://www.raspberrypi.org/archives/1892

desarrollo basado en este dispositivo es incluir, en alguna parte de este, el siguiente lema: *"Powered by Raspberry Pi"*. La Fundación no solicitará parte alguna de los beneficios generados por las ventas de ningún producto.

Para ayudarnos con la creación de nuevos proyectos los esquemáticos de los diferentes circuitos serán accesibles desde la misma página[93] de la fundación.
Además, Broadcomm liberó el código fuente del firmware del integrado. Esto permitirá a desarrolladores optimizar sus productos para este dispositivo.

Como observamos, no existe ninguna norma que nos impida utilizar esta placa para cualquier proyecto, aunque este implique la venta de un producto finalizado.

CubieBoard.

Cubietech Ltd. intenta motivar al usuario a que realice sus propios productos basándose en sus desarrollos. Así, los diseños de sus dispositivos son públicos y están alojados en su propia wiki (soloforce, 2013)[94].
Estos están publicados con una licencia del tipo *Creative Commons Attribution-Share Alike*[95]. Podremos usarlos para diseños comerciales siempre y cuando publiquemos nuestros productos con la misma licencia.

BeagleBoard.

La BeagleBoard Foundation, al contrario que el anterior ejemplo, regula el uso de sus productos en desarrollos comerciales. Para poner a la venta algún prototipo realizado con sus placas de desarrollo tendremos que contactar con la fundación.

Aun así, aunque este grupo es el más restrictivo de los mencionados en esta sección, los diseños de sus productos seguirán siendo públicos y de libre acceso. Accederemos a ellos desde la sección de soporte (pelochino.myopenid.com, 2013)[96] de su sitio web.
Los contenidos de su web y sus productos están bajo una licencia *Creative Commons Attribution-Share Alike*[97] esta nos permitirá usarlos para fines comerciales, siempre y cuando se mencione la fuente y se distribuyan con esta misma licencia.

Olinuxino.

Al referirnos a las licencias que utiliza Olimex Ltd tendremos que diferenciar entre las que aplica a su software y las que aplica al hardware.

---

[93] http://www.raspberrypi.org/technical-help-and-resource-documents
[94] http://docs.cubieboard.org/resources
[95] http://creativecommons.org/licenses/by-sa/3.0/
[96] http://beagleboard.org/Support/Hardware%20Support
[97] http://creativecommons.org/licenses/by-sa/3.0/

Con respecto a su hardware todos sus diseños y productos se encuentran bajo una licencia del tipo *Creative Commons Attribution-Share Alike*[98].
Al igual que en los puntos anteriores podremos usar estos productos para nuestros proyectos siempre y cuando mencionemos el origen y mantengamos la misma licencia en nuestros desarrollos.

El software se encuentra bajo una licencia diferente y es que la licencia utilizada para registrar sus desarrollos es del tipo GNU-GPL (Free Software Foundation, 2014)[99]. Esta licencia nos permitirá, al igual que la anterior, utilizar el código para nuestros intereses citando la autoría y manteniendo la licencia original en nuestro desarrollo.

Tanto los esquemáticos de los dispositivos como el software que montan de serie se podrán conseguir desde el perfil[100] de GitHub de Olimex Ltd. Hay que reconocer a esta empresa el esfuerzo por liberar todo el material de diseño de sus productos. En su perfil no solo encontraremos los circuitos electrónicos también podremos encontrar los archivos CAD utilizados para el diseño de las placas.

Galileo.

Galileo, al ser concebido como una colaboración entre Intel y Arduino, utilizará la licencia de este último. Esta será la misma que en los ejemplos anteriores, es decir, del tipo *Creative Commons Attribution-Share Alike*[101].
Como es habitual podremos usar este dispositivo para los usos que deseemos manteniendo el mismo tipo de licencia en nuestro desarrollo.

Conclusión.

Como observamos, todos los diseños anteriores tienen una licencia que nos permite su uso en nuestro proyecto. Nuestro criterio de selección será descartar aquellos proyectos que nos pueden dar algún problema en el futuro. Aquellos desarrollos cuya venta pueda ser problemática.

Así descartaremos las placas de la familia BeagleBoard por tener la obligatoriedad de notificar de nuestro desarrollo a la fundación que las produce. También eliminaremos de nuestra lista a la placa de prototipado Galileo. Y es que esta placa, aunque útil, no tiene una política de licencias muy clara, lo cual nos podría dar problemas en un futuro.

---

[98] http://creativecommons.org/licenses/by-sa/3.0/
[99] https://www.gnu.org/copyleft/gpl.html
[100] https://github.com/OLIMEX/OLINUXINO
[101] http://creativecommons.org/licenses/by-sa/3.0/

Elección final.

Crearemos una tabla con las características más interesantes de los preseleccionados.

	Características técnicas	Apoyo por parte de usuarios y empresa	Licencias utilizadas
Raspberry Pi Model B	Procesador ARM11. 512 MB de RAM.	500.000 mensajes en su foro principal. Numerosos blogs y foros amateur.	Licencia Open-Source.
BeagleBone Black	SOC AM335x ARM® CORTEX™ A8. 512 MB de RAM.	Listado de proyectos amateur. Extensísima wiki oficial.	Creative Commons Attribution-Share Alike.
A10-OLinuXino-LIME	SOC A10 ARM® CORTEX™ A8. 512 MB de RAM.	Pequeña comunidad amateur. Esquemas y diseños oficiales.	Creative Commons Attribution-Share Alike.

Tabla 4. Características de los modelos seleccionados.

Aunque, como podemos comprobar, la placa de prototipado Raspberry Pi Model B cuenta con las peores características técnicas a vistas de nuestro desarrollo será la más interesante.
Y es que su gran comunidad para resolver dudas la postula como la opción más aconsejable para un primer prototipado y una prueba de utilidad pública del prototipo.
La falta de potencia no será inconveniente en un proyecto que no demanda una gran capacidad de cálculo.

Elección de la interfaz de control.

El hecho de tener solo cuatro opciones disponibles facilitará la elección de un dispositivo. Al igual que en el epígrafe anterior desgranaremos una a una las características de los modelos preseleccionados.

Al terminar este punto, con todos los datos sobre la mesa, podremos emitir un juicio dictaminando que sistema se ajusta más a nuestras demandas.

Comparativa de características.

En la siguiente tabla resumiremos las características de los dispositivos mencionados en el anterior capítulo.

Modelo	FODTrack	LVB Tracker	LVB Tracker 2	WRAPS
Conector paralelo	Sí.	No.	No.	No.
Conector RS232	No.	Sí.	Sí.	No.
Conector USB	No.	Sí.	Sí.	Sí.

Conector RJ-45	No.	Sí.	Sí.	No.
Conector LCD	No.	Sí. Display HD44780.	Sí. Display HD44780.	Sí.
Yaesu GS-232	Sí.	Sí.	Sí.	No.
EasyComm	No.	Sí.	Sí.	Sí.
SatPc32	No.	No.	No.	Sí.
SAEBreTrack	No.	No.	No.	Sí.
Control manual	No.	No.	Sí.	Sí.
Funcionamiento autónomo	No.	No.	Sí.	No.

Tabla 5. Características de las interfaces de control.

Hemos omitidos algunos datos el factor de forma del dispositivo o el microcontrolador de los dispositivos ya que no son relevantes a efectos de funcionamiento de los mismos.

Comparativa de rendimiento.

Una comparativa de rendimiento como la realizada en el punto anterior no tendrá mucho sentido en este caso. Tres de los cuatro sistemas estudiados no están diseñados para llevar a cabo ningún cálculo que implique la necesidad de un microprocesador potente.
Aun así, sigue siendo interesante remarcar que el sistema *LVB Tracker 2* es capaz de funcionar de manera autónoma. Las funcionalidades del dispositivo serán las mencionadas en el anterior capítulo.

Comparativa de licencias.

Al ser este un tema de una naturaleza más compleja no usaremos una tabla para ilustrar los pormenores de cada producto. Punto a punto explicaremos bajo que licencias se encuentra cada sistema.

FODTrack.

Creado en una época en la que no existía el panorama open source actual, este dispositivo no fue dotado de ninguna licencia en su nacimiento.

Revisando la documentación proporcionada por el creador podremos encontrar lo más cercano a una limitación en su uso.
En el archivo *FTMANUAL.TXT,* en el apartado Copyright, se encuentra la siguiente frase:
- "FodTrack is free for noncommercial use. If you want to reward me somehow, write a piece of useful software and put it in the public domain!".
La traducción de la cita podría ser la siguiente:
- "FodTrack es gratis para un uso no comercial. Si quieres recompensarme de alguna manera escribe un software útil y ponlo a disposición del público".

Lo cual nos da a entender que podremos usar este sistema para pruebas pero que no podremos comercializarlo.

LVBTracker.

Aunque en la página web del desarrollador de este dispositivo se menciona que este se encuentra bajo una licencia open source, en la documentación del producto no se especifica bajo qué tipo de licencia está registrado.

LVBTracker 2.

La actualización de *LVBTracker* sí que fue dotada de una licencia específica.
En el código fuente del desarrollo disponible en la página web del creador podemos encontrar un párrafo refiriéndose a ella.
Acorde a este párrafo este desarrollo se encuentra bajo una licencia GNU General Public License en una versión 2.0 (Free Software Foundation, 2014)[102] o, en su defecto, en una versión superior.

Esta licencia nos permite el uso de este desarrollo siempre y cuando mencionemos la fuente y mantengamos el mismo tipo de licencia. El uso comercial, que no privativo, de este producto no está prohibido.

WRAPS.

Actualmente en desarrollo este dispositivo no posee licencia alguna. Aun así, atendiendo a la naturaleza educativa del proyecto, el creador anima el libre uso de su dispositivo en entornos no comerciales.

Elección final.

De las opciones anteriores la opción más interesante será LVB Tracker. Su simplicidad no será obstáculo en nuestro ya que la carga de cálculo la realizará la placa de control, no la interfaz de conexión.

[102] http://www.gnu.org/licenses/gpl-2.0.html

4 – Componentes.

Introducción.

Para facilitar su locura este capítulo estará dividido en dos secciones principales.

En la primera de ellas, *Hardware*, realizaremos un detallado repaso de las características técnicas y los campos de aplicación de los dispositivos elegidos.
Dejaremos para la segunda parte el análisis del software que usaremos en nuestro desarrollo. Dentro del análisis incluiremos el proceso de instalación y configuración de este.

Hardware.

Los tres sistemas principales que compondrán nuestro proyecto serán: la placa de prototipado, la interfaz de conexión y el conversor UART-TTL a RS-232.

Raspberry Pi.

Ya detallamos en anteriores capítulos las características generales de este dispositivo. Ahora nos tocará entrar en los detalles técnicos.

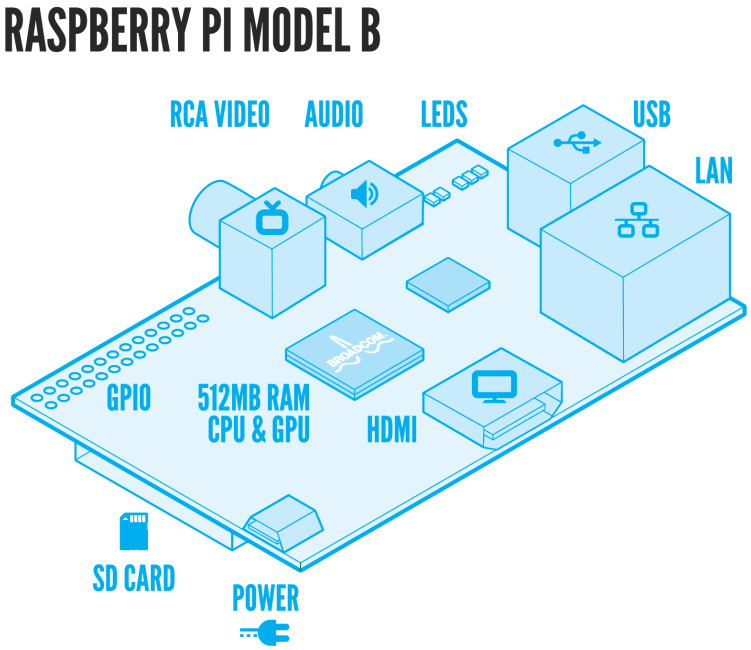


Figura 25. Croquis de la placa de desarrollo. [103]

[103] http://www.raspberrypi.org/faqs

Potencia de cálculo.

El microprocesador y el procesador gráfico se encuentran en un mismo integrado, el chip Broadcom BCM2835.

Como comentamos en el anterior capítulo, este integrado contiene un microprocesador ARM1176 y un procesador gráfico VideoCore IV de doble núcleo.
Este microprocesador, aunque veterano, es capaz de correr con un entorno de escritorio completo. Las capacidades del chip gráfico serán más que suficientes, pudiendo manejar según su sitio web oficial, videos con una resolución FHD.

El microprocesador funcionará por defecto a 700 megahercios aunque se podrá overclockear sin problemas a 800 megahercios. Velocidades más altas requerirán del uso de disipadores.
En un principio, para asegurar la estabilidad del sistema, mantendremos las frecuencias y tensiones por defecto del dispositivo.

Las capacidades gráficas del dispositivo estarán desaprovechadas en un principio. No implementaremos ninguna programación en OpenGL.

Tipos de memoria.

Hablando sobre las diferentes clases de memoria del dispositivo podremos mencionar que la memoria RAM es de 512 megabytes. Al encontrarse dentro del integrado BCM2835 no será posible ampliarla. Por el mismo motivo, la memoria está compartida entre el sistema operativo y el chip gráfico.
La cantidad de memoria asignada al microprocesador y al chip gráfico se podrá ajustar sin ningún problema.

La memoria no volátil se encontrará en la tarjeta de memoria SD y en los dispositivos conectados por el puerto USB. Las tarjetas SD tendrán que ser de clase 4 o superior. Se recomienda una tarjeta de clase 10 para un rendimiento óptimo del sistema.

Para que el sistema pueda arrancar correctamente, la partición de arranque de nuestro sistema operativo (/boot) tendrá que alojarse en la tarjeta de memoria. Ahí es donde acudirá el bootloader al encender el sistema.

Durante el desarrollo del proyecto se estudiará la conveniencia de usar un disco duro externo o una memoria flash para almacenar los datos del usuario.

Conexiones de video.

Aunque el sistema dispone de salida RCA y HDMI, durante la mayor parte de este proyecto utilizaremos el terminal para realizar las pruebas de funcionamiento.

La diferencia de la calidad del video proporcionado por una y por otra salida será más que evidente. Mientras que conectando nuestra placa a un monitor, o televisión, mediante el puerto HDMI obtendremos una imagen con resolución FHD (1920x1080) el puerto RCA solo nos podrá proporcionar una salida con calidad PAL o NTSC.

Tendremos que mencionar también que podremos conectar un conversor de vídeo a la interfaz HDMI para obtener una salida en formato VGA.
Cuando el sistema esté completado el control se podrá realizar desde una pantalla conectada al puerto GPIO del dispositivo.
Se analizarán los pros y los contras de la utilización de una pantalla táctil para el control del programa de guiado.

Leds de control.

Los diversos leds de la placa nos proporcionarán información sobre diferentes factores como pueden ser alimentación, conexión a redes y acceso a memoria.
La situación de estos podremos comprobarla en la siguiente imagen.

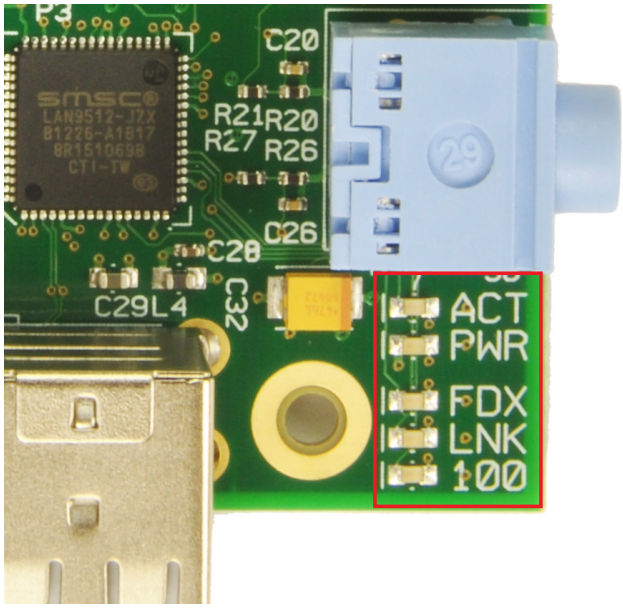


Figura 26. Leds de estado de Raspberry Pi (Kar, 2013)[104]

El significado de éstos será el siguiente:

- "ACT". Se iluminará en verde si el acceso a la tarjeta de memoria es correcto.
- "PWR". Si el sistema está alimentado a la tensión y corriente correcta el led emitirá una luz roja.
- "FDX". La conexión Ethernet se encuentra activa cuando este led verde se encienda.

[104] http://www.rpiblog.com/2012/12/raspberry-pi-status-indicator-led-info.html

- "LNK". Cuando la conexión cableada emita o reciba paquetes de datos se iluminará con el color verde.
- "100". La velocidad de la red cableada es de 100 megabits si este led emite una luz amarilla.

El sistema identifica estos LEDS como puertos de salida GPIO. Esto nos posibilitará configurar la manera en la que actúan.

Salida de audio.

La conexión para la salida de audio no tendrá ningún uso en este proyecto.
La deshabilitaremos para reducir el consumo del sistema.

Alimentación.

Esta placa está alimentada por el puerto microUSB. Para un correcto funcionamiento del sistema tendremos que usar una fuente de alimentación que nos proporcione una tensión de 5 voltios y una corriente mínima de 700 mA.

Este valor mínimo no será válido si conectamos al sistema dispositivos que consuman una gran cantidad de corriente. Así, si usamos adaptadores conectados por los puertos USB, necesitaremos una corriente de entrada mayor. Para no tener ningún problema por falta de alimentación se recomienda utilizar una fuente de corriente de, al menos, 1 A.

LVB Tracker.

La placa LVB Tracker nació como proyecto en el coloquio británico de la AMSAT del año 2003. Como hemos comentado anteriormente, pretende ser una alternativa asequible a controladores como el Yaesu GS-232.

Ya dimos un somero repaso a las características de esta interfaz de conexión en el anterior capítulo.
Detallaremos punto a punto las peculiaridades de este sistema.

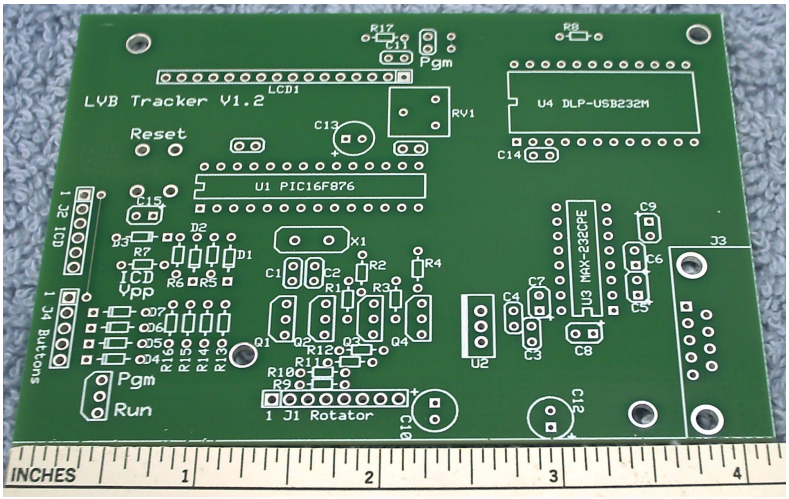


Figura 27. Placa LVB Tracker V1.2 sin componentes (AMSAT, 2014)[105].

Potencia de cálculo.

Para el procesado de las señales provenientes del sistema de control utiliza un microcontrolador PIC 16F876 o 16F876A.

El sistema está preparado para programarse sin necesidad de ningún otro dispositivo. El creador del mismo da la opción de utilizar el puerto RS-232 o el puerto ICD.
El proceso de programación del microcontrolador será diferente dependiendo del tipo de conexión que utilicemos.
Si deseamos utilizar el puerto RS-232 tendremos que puentear las posiciones indicadas como *PROGRAM* antes de realizar ninguna acción. Después de esto podremos proceder con la programación del integrado.
En el supuesto de que prefiramos el puerto ICD, las posiciones de los puentes estarán en el modo *RUN*. La otra peculiaridad de este método será que tendremos que desconectar el condensador número C16.

Aunque disponible, el desarrollador de este proyecto aconseja utilizar el puerto RS-232 antes que el puerto ICD.

El código para programar el microcontrolador está disponible en la misma página web del desarrollo.

Conexiones.

Uno de los puntos fuertes de este sistema es la variedad de puertos de entrada de los que dispone. El dispositivo está preparado para recibir órdenes a través de un puerto USB, RS-232 o Ethernet. La única limitación vendrá de la obligatoriedad de usar solo una interfaz al mismo tiempo.

---

[105] http://store.amsat.org/catalog/images/2012%20LVBTrackerBoard%20L.jpg

Cada puerto necesitará de un hardware específico para su funcionamiento.

El puerto USB se valdrá de un integrado FT232BM comercializado por la compañía FTDI. LVB Tracker está diseñado para poder alojar un módulo de evaluación DLP-USB232M el cual dispone del anterior integrado. Para hacer uso de este solo habrá que soldar el modulo al socket del diseño.

La disponibilidad del puerto serie será posible a través de un integrado MAX-232CPE. Este se encargará de la conversión de los niveles de tensión de la señales del microcontrolador. Cinco condensadores (C5#, C6#, C7#, C8# y C9#) serán necesarios para el funcionamiento de este dispositivo.

El puerto Ethernet funcionará mediante un integrado de la compañía LANTRONIX.
Las diferentes tensiones en el circuito obligarán al uso de convertidores de tensión. Éstos harán necesario la instalación de disipadores para un correcto funcionamiento del conjunto.

Elección de una conexión de control.

Vistas las tres opciones anteriores, tendremos que elegir una en concreto para nuestro diseño.

Por la simplicidad y el bajo coste optaremos por utilizar el puerto serie RS-232.

Además esta elección nos facilitará la conexión a la placa de control. Raspberry Pi dispone de un puerto UART-TTL que, a través de un simple convertidor, podremos transformar en un puerto RS-232.

Acorde a las instrucciones del creador del desarrollo, este puerto también nos facilitará la labor de programar el microcontrolador.

Visualización de datos.

La placa también posee conectores para un LCD de dos elementos. Éste nos permitirá visualizar los valores actuales de la posición del rotor.
Al tener un menor consumo de corriente puede ser una opción interesante para el control del dispositivo cuando no haga falta mayor funcionalidad que cambiar el acimut y la altura de la antena.

Botones de control.

Para operar el sistema de manera autónoma el diseño de esta placa está preparado para poder mover el rotor mediante cuatro botones. Estos botones corresponden a las cuatro opciones básicas, esto es, arriba, abajo, izquierda y derecha.

Además de utilizarse para mover el sistema a voluntad son necesarios para las labores de calibrado del conjunto.

Conversor de niveles TTL a niveles R-232.

Como mencionamos anteriormente para conectar nuestra placa de control a la interfaz de conexión necesitaremos un convertidor de señal. Para este cometido utilizaremos el ampliamente usado MAX3232.

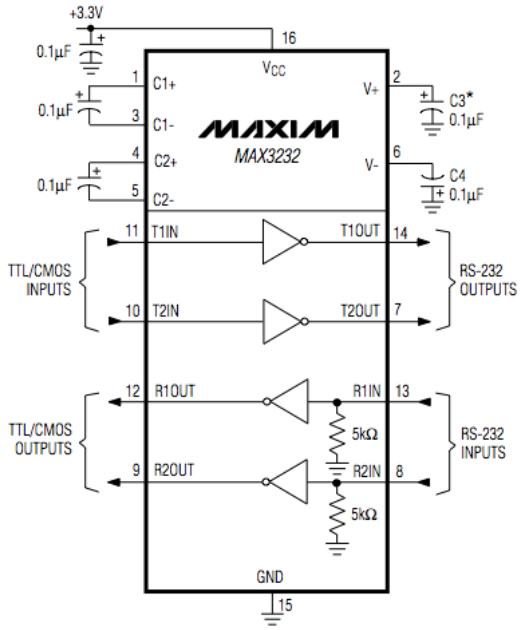


Figura 28. Circuito típico de utilización (Maxim Integrated Products ©, 2007)[106].

El MAX3232 es una evolución del clásico MAX232. Este nuevo integrado soporta un mayor rango de tensiones de entrada y unas velocidades de transferencia mayores.

En el capítulo séptimo abordaremos la creación del conversor detallando el funcionamiento de este integrado.

Software.

En este punto trataremos la instalación de las librerías, los programas y los sistemas operativos que necesitaremos en nuestro desarrollo.
Abordaremos primero la instalación del sistema operativo de la placa de prototipado. Una vez instalado éste explicaremos paso a paso como configurarlo para nuestros intereses.

---

[106] Página 12.

Incluiremos también la instalación y configuración de Python necesaria para ejecutar nuestro script en Windows y Mac OS X.

Raspbian.

El sistema operativo que utilizaremos en la Raspberry Pi de nuestro desarrollo será Raspbian en su última versión disponible. Esta versión, a fecha de hoy, es la publicada el día veinticinco de septiembre de este mismo año. Se puede descargar desde su web oficial (The Raspberry Pi Foundation)[107].

Raspbian es un port de Debian optimizado para la Raspberry Pi. El sistema operativo original ha sido modificado para incluir las librerías necesarias para interactuar con el hardware del dispositivo. Aunque sigue el ritmo de actualizaciones de Debian, Raspian posee sus propios repositorios.

Instalación del sistema operativo.

Para instalar el sistema operativo tendremos que recurrir a alguno de los dos métodos disponibles.

El primero de ellos consiste en, mediante alguna utilidad, copiar en una tarjeta de memoria el conjunto de directorios del sistema operativo con todos los ficheros necesarios. Así podremos ejecutar el sistema operativo elegido al introducir esta tarjeta en nuestra Raspberry Pi.

Aunque el método anterior no revierte una gran dificultad, la Raspberry Pi Foundation creó una utilidad para mejorar la gestión de los sistemas operativos instalados en una tarjeta de memoria.

Esta utilidad, NOOBS (liz, Introducing the New Out Of Box Software (NOOBS) | Raspberry Pi, 2013)[108], nos permite instalar varios sistemas operativos en una misma tarjeta. Una vez instalados podremos iniciar cualquiera de ellos configurando antes, si así lo deseamos, parámetros como idioma por defecto, distribución de teclado o formato de salida de vídeo.

Utilicemos el procedimiento que utilicemos para preparar nuestra tarjeta de memoria tendremos que tener en cuenta la velocidad de esta. Para que el sistema se comporte de una forma fluida será necesario que la memoria sea, al menos, de una clase cuatro.
También será importante la capacidad de ésta y es que no es recomendable que sea superior a 4 GB.

---

[107] http://www.raspberrypi.org/downloads
[108] http://www.raspberrypi.org/archives/4100

En nuestro proyecto desecharemos la utilización de NOOBS ya que solo usaremos un sistema operativo en nuestro desarrollo. Obviaremos pues la instalación y configuración de NOOBS y nos centraremos en el resto de métodos disponibles.

Linux.

En un sistema Linux podremos realizar la instalación desde la línea de comandos o desde un programa creado para este cometido. Analizaremos a continuación cada caso.

Línea de comandos.

Los pasos necesarios serán los siguientes:

1. Ejecutar desde la terminal el comando "`df –h`". Así sabremos que unidades están conectadas.
2. Conectar nuestra memoria SD.
3. Ejecutar de nuevo la orden "`df –h`". Comprobaremos que nueva unidad ha aparecido en el listado.
4. Desmontar la unidad que ha aparecido, nuestra tarjeta, mediante la siguiente orden: "`umount /dev/sdd1`". Donde sdd1 será el nombre de nuestra memoria.
5. Copiar la imagen de Raspbian a la tarjeta utilizando el comando dd. Usaremos la siguiente orden: `dd bs=4M if=sistemaoperativo of=tarjeta`
   "sistemaoperativo" será la localización de nuestra imagen y "tarjeta" será la unidad de nuestra memoria.
6. Una vez terminada la operación el terminal volverá a activarse de nuevo.
7. Desmontar la tarjeta repitiendo el paso cuarto.

Siguiendo estos pasos la tarjeta estará lista para ser usada en nuestra Raspberry Pi.

Interfaz gráfica.

La única opción en este campo que menciona la wiki de Raspberry es una aplicación llamada ImageWriter[109].

[109] https://apps.ubuntu.com/cat/applications/precise/usb-imagewriter/

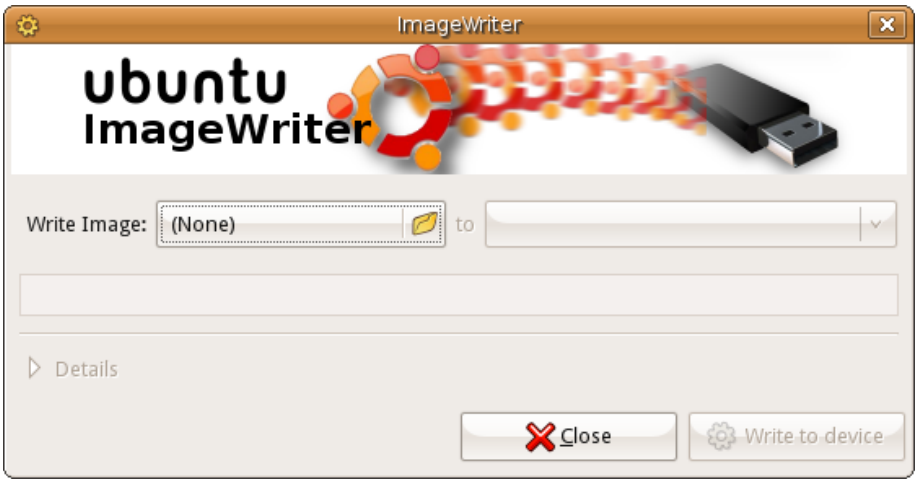


Figura 29. Ventana principal de ImageWriter.

Este programa únicamente nos permitirá copiar la imagen de nuestro sistema operativo en la tarjeta elegida. No dispone de opciones para realizar copias de seguridad.

Disponible solo para Ubuntu, se podrá instalar desde su propio Centro de Software.

Mac OS X.

El proceso de creación de la tarjeta SD en un sistema operativo Mac OS X compartirá algunos métodos con los disponibles para Linux. Analizaremos, como en el ejemplo anterior, cada uno de los procedimientos disponibles.

Línea de comandos.

En este caso, acorde a la wiki[110] oficial, tendremos varias maneras de solucionar este problema. Explicaremos cada procedimiento por separado para facilitar su comprensión.

Método primero.

Los pasos a seguir serán los descritos en la página mencionada anteriormente. Estos serán los siguientes:

1. Ejecutar "`diskutil list`" desde el terminal de Mac OS X. Así podremos identificar el nombre de nuestra unidad del listado.
2. Desmontar la unidad mediante el comando "`diskutil unmountDisk /dev/disk`". Reemplazaremos disk con el nombre de nuestra unidad.

110 http://elinux.org/RPi_Easy_SD_Card_Setup#Using_system_tools_.28mostly_graphical_interface.29

3. Ejecutar de nuevo la orden dd. La sintaxis será: "`sudo dd bs=1m if=sistemaoperativo of=tarjeta`". Como en el ejemplo anterior sistemaoperativo será la localización de la imagen a grabar y tarjeta se corresponderá con la localización de la unidad de memoria.
4. Una vez terminada la operación, que puede llevar bastante tiempo, expulsaremos la tarjeta SD repitiendo el segundo paso.

Método segundo.

Con algunas similitudes con respecto al anterior procedimiento, para realizar este tendremos que llevar a cabo las siguientes acciones:

1. Ejecutar la acción "`df –h`" desde el terminal. Así obtendremos un listado de las unidades conectadas al ordenador.
2. Conectar la tarjeta de memoria.
3. Utilizar de nuevo el comando "`df –h`". Nos fijaremos en que unidad está ahora en el listado que no estaba antes. Esta unidad será nuestra tarjeta SD.
4. Desmontar nuestra unidad. Para ello utilizaremos la siguiente orden: "`sudo diskutil unmount /dev/disks1`". En lugar de `disks1` utilizaremos el nombre obtenido en el paso anterior.
5. Transformar el nombre obtenido en el tercer paso. Esto se hará para obtener el nombre de la unidad completa y no el de una partición en concreto. Para ello tendremos que eliminar la terminación "`sX`", donde `X` es un número cualquiera y añadir como prefijo la letra "`r`" `a nuestra unidad`.
6. Ejecutar el comando dd para copiar la imagen. Como en el caso precedente el comando será: "`sudo dd bs=1m if=sistemaoperativo of=/dev/tarjeta`". En "sistemaoperativo" irá la imagen a copiar y en "tarjeta" el nombre obtenido en el quinto paso.
7. Una vez completado el último paso expulsaremos la tarjeta mediante la siguiente acción: "`sudo diskutil eject /dev/tarjeta`".

Interfaz gráfica.

Es en los programas con interfaz gráfica donde encontraremos diferencias. En la página de la wiki oficial de Raspberry dedicada a este tema encontraremos el siguiente listado de aplicaciones:

- ApplePi-Baker
- PiWriter
- Pi Filler

Todas estas aplicaciones nos permitirán crear tarjetas de memoria con imágenes de NOOBS o de cualquier sistema operativo disponible para Raspberry Pi. Algunos de los programas mencionados anteriormente tendrán más funcionalidades como podrá ser la realización de copias de seguridad o la asignación de formato a las unidades de memoria.

A continuación realizamos un repaso somero a las aplicaciones mencionadas.

ApplePi-Baker

Además de la grabación de sistemas operativos en nuestra tarjeta SD, ApplePi-Baker nos ofrece la oportunidad de realizar copias de seguridad de nuestra tarjeta de memoria.

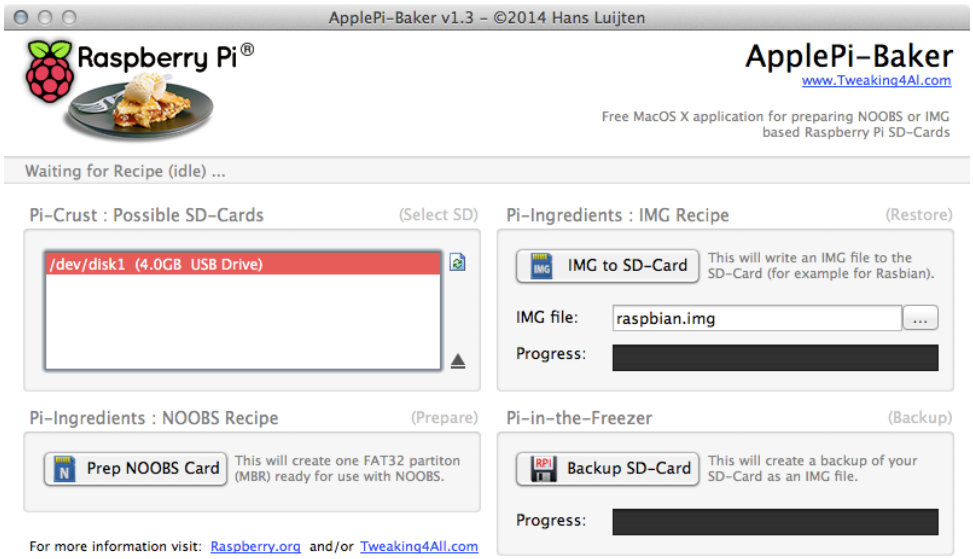


Figura 30. Ventana principal de la aplicación ApplePi-Baker.

PiWriter

Concebido para la grabación del sistema operativo esta aplicación está diseñada para evitar problemas con la elección de la tarjeta.

Para cumplir su cometido seguirá la misma metodología que los procedimientos vistos con anterioridad para la línea de comandos. El programa hará, de manera transparente al usuario, un listado de las unidades disponibles antes y después de introducir la tarjeta de memoria en nuestro sistema.
Hecho esto nos mostrará su opción más correcta y nos pedirá que confirmemos su elección antes de realizar cualquier acción.

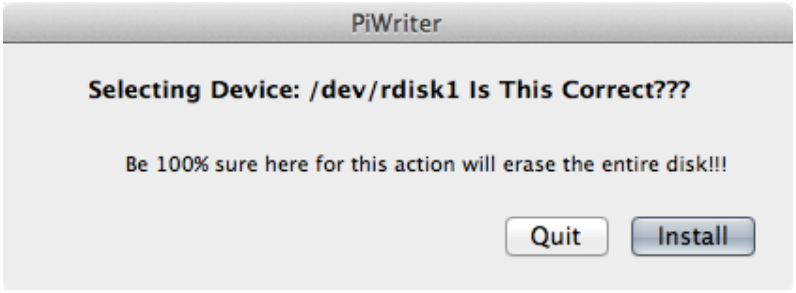


Figura 31. Diálogo de comprobación de PiWriter.

El tiempo que esta aplicación necesitará para llevar a cabo las operaciones de formateo de la tarjeta y grabación del sistema de ficheros dependerá de las características de la tarjeta de memoria y del sistema operativo a grabar.

Para realizar las tareas descritas anteriormente tendremos que tener derechos de administrador del sistema.

Pi Filler

Una vez autentificados como administradores del sistema esta aplicación realizará una comprobación de las unidades disponibles antes y después de conectar nuestra tarjeta SD. De esta manera podrá reconocer nuestra tarjeta de memoria.

A continuación Pi Filler nos pedirá que confirmemos su elección.

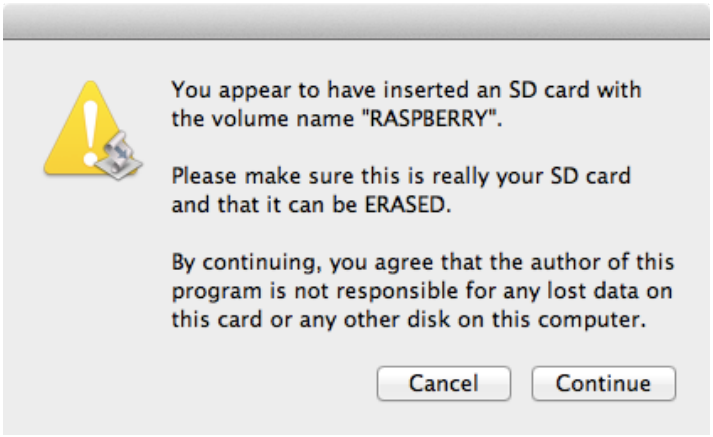


Figura 32. Diálogo de comprobación de Pi Filler.

Para evitar problemas, como el borrado accidental de archivos, el software nos volverá a pedir nuestra confirmación antes de iniciar el proceso de borrado de la tarjeta.

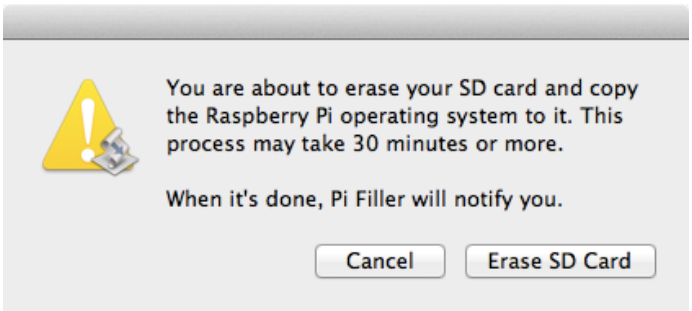


Figura 33. Diálogo de comprobación de Pi Filler.

Cuando finalice el proceso de formateo de la tarjeta de memoria y grabación del sistema operativo la aplicación nos avisará mediante otro menú desplegable.

Windows.

Sólo dos opciones estarán disponibles para los sistemas operativos de la familia Windows. Con respecto a las opciones vistas en los puntos anteriores éstas harán uso de herramientas distintas debido a las diferencias existentes entre los sistemas operativos.

En ambas posibilidades el proceso de reconocimiento de la tarjeta de memoria dependerá del usuario. Al contrario que las aplicaciones y métodos vistos anteriormente estas aplicaciones no realizaran ningún reconocimiento de las unidades conectadas. Sin embargo, el proceso de creación del sistema de ficheros también tendrá que realizarse con privilegios de administrador del sistema.

A continuación analizaremos las aplicaciones mencionadas.

Win32 Disk Imager.

La primera opción de la que hablaremos cuenta con una interfaz gráfica para facilitar el proceso de creación de la tarjeta de memoria.

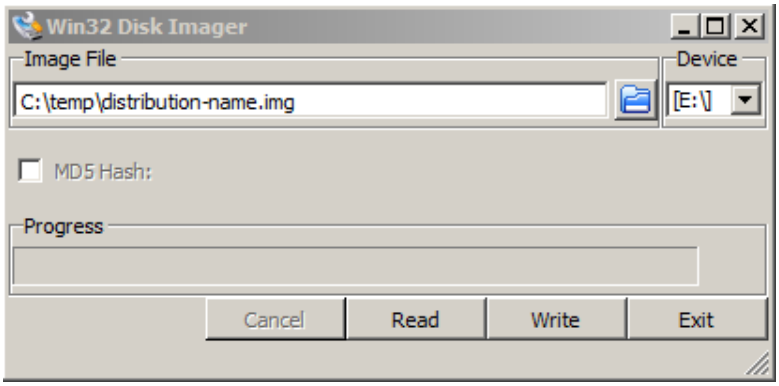


Figura 34. Ventana principal de Win32 Disk Imager.

Al obviar el proceso de selección de la unidad los pasos a seguir serán los siguientes:

1. Introducir la tarjeta de memoria en nuestro lector.
2. Ejecutar este programa con privilegios de administrador. Si no estamos utilizando una cuenta de administrador tendremos que ejecutar el programa con el botón derecho pulsando sobre "Ejecutar con privilegios de administrador".
3. Seleccionar la imagen del sistema operativo a grabar.
4. Seleccionar la unidad asignada a la tarjeta de memoria.
5. Pulsar en el botón "Write".

Esta aplicación nos dará la opción de comprobar el hash del fichero para verificar que la imagen del sistema operativo descargada es la correcta. Aunque no imprescindible es un paso recomendado para mejorar la seguridad de nuestro sistema.

flashnul.

La segunda utilidad funcionará desde la línea de comandos de Windows. Aunque menos cómoda, esta aplicación es recomendable en el supuesto de que la primera opción falle o no esté disponible.

Una vez descargado el sistema operativo que deseemos instalar los pasos a seguir serán los siguientes:

1. Ejecutar la línea de comandos con privilegios de administrador.
2. Introducir la siguiente orden "`C:/flashnul/flashnul.exe –p`".
3. Utilizar la orden "`-L`" para cargar el sistema operativo en la tarjeta de memoria. Para ellos utilizaremos la siguiente acción: "`C:/flashnul/flashnul.exe E: -L C:/temp/distribucion.img`"
4. Una vez realizada la instalación la aplicación nos avisará con un mensaje.

Conclusión.

Sirvan las guías anteriores como una primera aproximación a la instalación del sistema operativo en nuestro dispositivo de memoria. Todas las aplicaciones mencionadas anteriormente están disponibles en el disco óptico que acompaña a esta memoria.

En el supuesto de que queramos posteriormente ampliar información al respecto tendremos que acudir a la página de la wiki oficial dedicada a este tema o al foro oficial de la fundación.

Configuración inicial del sistema operativo.

Formateada y graba la tarjeta SD, procederemos a personalizar la configuración del sistema operativo creado. Esto será necesario ya que ciertos parámetros, como la distribución del teclado, el idioma por defecto y la zona horaria del sistema vendrán por defecto fijados para un usuario del Reino Unido.

La configuración inicial del sistema operativo se podrá realizar de dos maneras:

- Conectando el dispositivo a un monitor, teclado y ratón.
- Mediante otro ordenador, conectado a la misma red, realizando la configuración por SSH.

Por facilidad y por rapidez nos conectaremos a nuestro sistema mediante un túnel SSH.
Para ello tendremos que conocer la dirección IP que nuestro router ha asignado a nuestra Raspberry Pi. Nos conectaremos utilizando la línea de comandos (en Linux y Mac OS X) o mediante PuTTy en los sistemas operativos de la familia Windows.

En nuestro caso, en Mac OS X 10.9.2, desde el terminal ejecutaremos el siguiente comando:

```
1  ssh pi@192.168.1.14
```

Una vez dentro del sistema introduciremos el usuario y contraseña por defecto de Raspbian. Este usuario es pi y la contraseña será raspberry.

Ejecutaremos el script de configuración, `raspi-config`, obteniendo esta ventana.

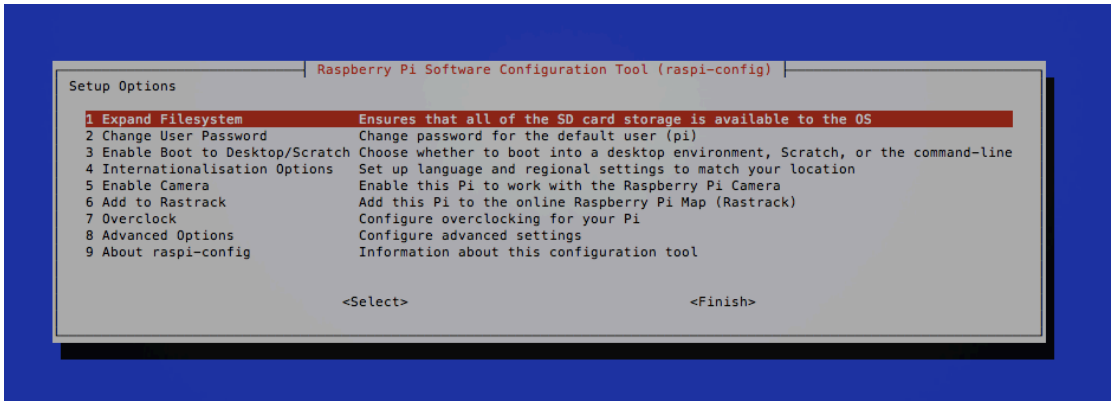


Figura 35. Ventana principal del script de configuración "raspi-config".

A continuación mostraremos las diferentes opciones que tendremos que cambiar para el correcto funcionamiento de nuestro desarrollo.

Opciones locales.

Dentro de este campo tendremos que ajustar el idioma por defecto, la distribución del teclado que usaremos y la zona horaria donde residamos. El sistema no posee un reloj de tiempo real, así que usará el desfase horario para ajustar la hora del sistema a partir de la hora universal obtenida de un servidor.

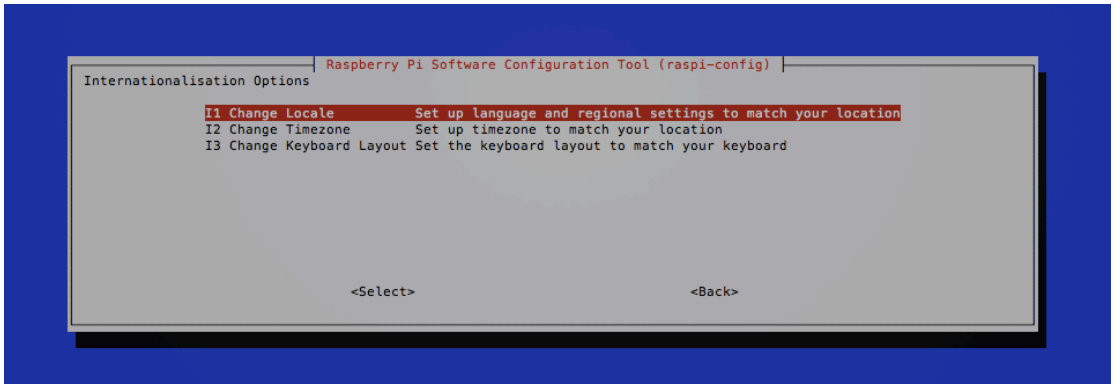


Figura 36. Menú de selección de opciones locales.

Idioma local.

Nuestra elección sobre un idioma local repercutirá en qué paquetes serán descargados al solicitar un fichero de los repositorios oficiales. El gestor de paquetes buscará, en primer lugar, los paquetes en el idioma seleccionado.

La cantidad de paquetes, y traducciones, disponibles es enorme, así que el número de idiomas de la siguiente lista será alto.

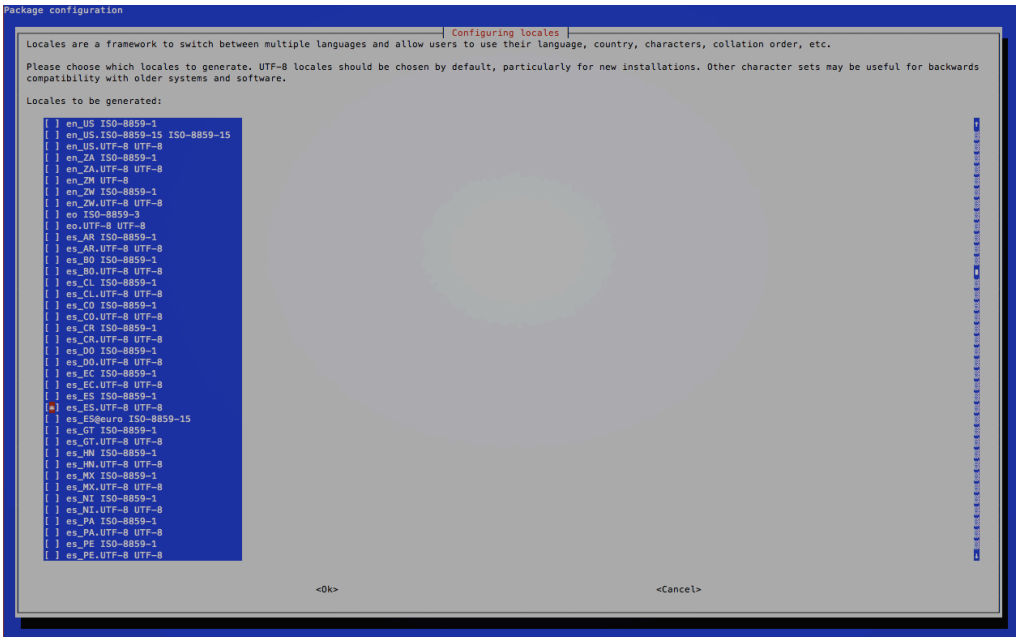


Figura 37. Selección de idioma del sistema.

Seleccionaremos es_ES.UTF-8. Esto es, idioma español de España y con el juego de caracteres UTF-8.

Zona horaria.

A través de varios menús en el script iremos definiendo nuestra posición.

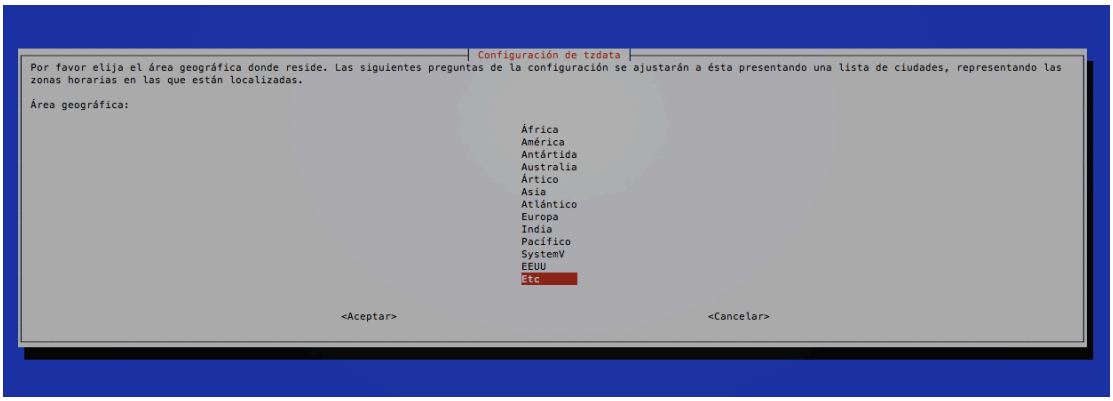


Figura 38. Menú de selección de la localización.

El script dispone de una base de datos con las zonas horarias de cada país, al especificar nuestra posición sabrá que desfase aplicar a la hora universal obtenida por la red.

Una vez definida la posición nos devolverá un breve mensaje recordándonos nuestra elección.

```
Current default time zone:'Europe/Madrid'
Local time is now:      Sun Mar 16 11:14:06 CET 2014.
Universal time is now:  Sun Mar 16 10:14:06 UTC 2014.
```

Teclado.

Elegir una distribución de teclado solo tendrá sentido en el caso de que deseemos utilizar nuestra Raspberry Pi con un monitor y un teclado, no mediante un túnel SSH.

En el supuesto de que nos conectemos mediante una conexión SSH el sistema interpretará las pulsaciones de teclado atendiendo a la distribución de teclado de nuestro sistema operativo.

Pip.

El módulo de cálculo utilizado en este proyecto no está instalado por defecto en Python. Para poder hacer uso de él tendremos que instalarlo mediante un gestor de paquetes.

El gestor de paquetes más usado para Python es pip. Según su guía oficial (PyPA, 2013)[111], en Debian, para instalar pip tendremos que ejecutar, con privilegios de administrador, la siguiente orden:

```
1  sudo apt-get install python-pip
```

Paquetes de desarrollador.

Nuestra distribución, Raspbian, no posee algunos paquetes de los que sí dispone Debian y tendremos que instalar más librerías para el correcto funcionamiento de nuestro desarrollo. Ejecutaremos la siguiente orden para instalar las librerías de desarrollador de Python.

```
1  sudo apt-get install python-dev
```

Ahora podremos pasar a la instalación de la librería de cálculo PyEphem.

[111] http://www.pip-installer.org/en/latest/installing.html

PyEphem.

Esta librería posee dos versiones. Una de ellas creada para Python 3 mientras que la otra es válida en Python 2. Ambas están identificadas con diferentes nombres, ephem será el desarrollo creado para Python 3 y pyephem estará adaptada a Python 2.

Ya que en nuestro desarrollo utilizaremos Python 2.7.6, la librería que utilizaremos será pyephem. Para instalarla tendremos que ejecutar el siguiente comando:

```
1  sudo pip install pyephem
```

Si todo ha ido bien obtendremos un mensaje de confirmación.

Windows.

En los sistemas operativos de la familia Windows (XP, Vista, 7, 8) no viene incluida ninguna distribución de Python. Tendremos que instalar la distribución deseada, en nuestro caso la versión 2.7.6.

Python se puede descargar desde la página web oficial (Python Software Foundation, 2013)[112].

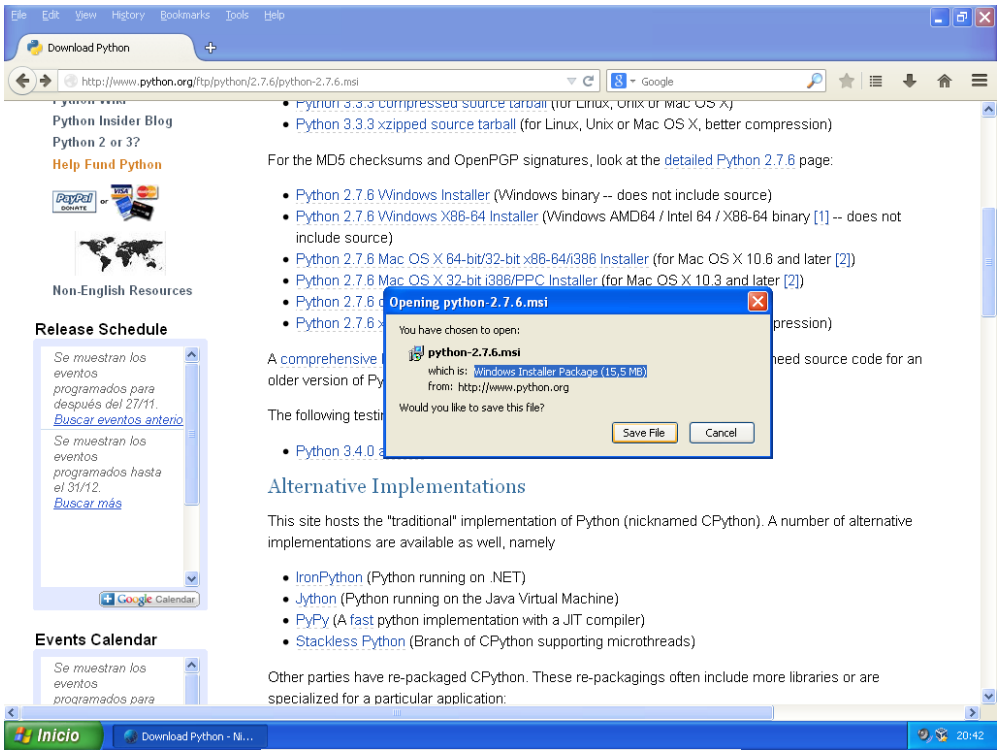


Figura 39. Descarga de Python en Windows XP.

El proceso de instalación no revestirá de ninguna dificultad.

[112] http://www.python.org/ftp/python/2.7.6/python-2.7.6.msi

Pip.

La instalación de un gestor de paquetes como pip en Windows presentará más dificultades que en los sistemas operativos basados en Linux.

Recurriremos a pip-Win. Un desarrollo libre que, en un mismo instalador, instala pip y virtualenv. Con una licencia del tipo GNU LGPL (Free Software Fundation, 2013)[113] podremos descargar esta aplicación desde el sitio[114] web de su creador.

Realizando la instalación de pip-Win en una instalación limpia de Windows XP obtendremos la siguiente ventana.

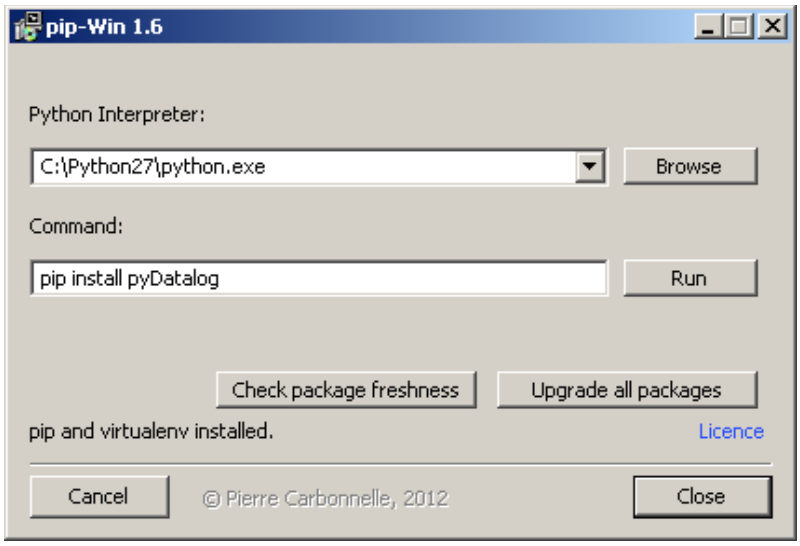


Figura 40. Instalación inicial de pip-Win.

La instalación de pip-Win también nos proveerá de las librerías de desarrollador necesarias para la compilación de paquetes en Python.

PyEphem.

La instalación de PyEphem se realizará mediante un ejecutable de Windows que podremos descargar desde la página (Rhodes, 2013)[115] web del desarrollador de este proyecto.

Tendremos ejecutables para las versiones 2.6 y 2.7 de Python.

---

[113] http://www.gnu.org/licenses/gpl.html
[114] https://sites.google.com/site/pydatalog/python/pip-for-windows
[115] http://rhodesmill.org/pyephem/

Mac OS X

Apple incluye en cada una de sus versiones de Mac OS X la última versión disponible de Python. En el sistema operativo desde el que trabajaremos, 10.9, la versión incluida de Python será 2.7.6.
Esta será perfectamente compatible con nuestro proyecto.

Pip

La instalación de pip en Mac OS X requerirá que descarguemos el script de instalación de la página (PyPA, 2014)[116] oficial de este desarrollo.

Ejecutaremos el script que hemos descargado para instalar este gestor de paquetes.

```
1  python get-pip.py
```

Cuando el script termine sus operaciones tendremos disponible pip y setuptools en nuestro sistema.

PyEphem

Con pip instalado la instalación de PyEphem no tendrá ninguna complicación. Para ello, desde la terminal de Mac OS X, ejecutaremos la siguiente orden.

```
1  sudo pip install pyephem
```

Si todo va bien y el sistema realiza las tareas sin ningún problema nos devolverá el siguiente mensaje:

```
1  Successfully installed pyephem
2  Cleaning up...
```

Si nos devolviera algún mensaje de error tendríamos que instalar las librerías que fueran necesarias.

[116] http://www.pip-installer.org/en/latest/installing.html

5 - GUI.

Debido a la gran extensión del código fuente del script dividiremos la explicación de este en diferentes capítulos.
Primero entraremos en la descripción de cómo se creó la interfaz gráfica del programa. Detallaremos los problemas que surgieron con la librerías utilizadas para la interfaz y como llegamos al diseño definitivo. Acompañaremos este recorrido con diferentes capturas de pantalla.
En el siguiente capítulo desmenuzaremos las clases relacionadas con el cálculo de la posición y los protocolos de conexión.

PyGTK, la primera elección para la interfaz.

La opción más viable en un principio parecía el uso de la librería PyGTK.
Esta librería es un binding de la librería original en C++ diseñada para el proyecto GTK. GTK es ampliamente conocida como la librería de desarrollo del proyecto GNOME.
El uso de una librería del proyecto GNOME garantizaba el cumplimiento de los principios del software libre.
PyGtk está soportado por una licencia del tipo LGPL.

Si queremos más información sobre las licencias de tipo LGPL podemos consultar su sección oficial en la web del proyecto GNU (Free Software Fundation, 2013).[117]

La referencia más importante para el uso de PyGTK son los manuales que contiene su página web oficial.[118]

Comencé el diseño de la interfaz principal utilizando el último manual de la versión 2.4 (Finlay, 2012).

Los principios para el diseño iban a ser los siguientes:

1. Pequeño tamaño de interfaz y programa.
2. Funcionamiento sencillo
3. Facilidad de uso

La futura utilización del programa en una pequeña pantalla obligaba al que la ventana principal del programa fuera de pequeño tamaño.
El tamaño del script y la optimización de éste tendrían que ser lo suficientemente buenos para poder ejecutarlo, en un principio, en una Raspberry Pi.

[117] http://www.gnu.org/licenses/lgpl-3.0.html
[118] http://www.pygtk.org/

El programa tendría que poder usarse por cualquier usuario. Aunque tuviera disponibles opciones más complejas, para usuarios más avanzados, cualquier usuario inexperto debería poder usarlo.
Seleccionar localización, seleccionar objeto y activar seguimiento. No tendría que dar más problemas.

La posible utilización del dispositivo en ambientes adversos fuerza al dispositivo a ser fácil de usar.
Botones grandes, facilidad para usarlo con una pantalla táctil o con una rueda de selección.

Elementos de la ventana principal

En la ventana principal serán necesarios estos elementos:

- Menú de selección de la localización
- Menú de selección de la familia del satélite
- Menú de selección del satélite de la familia elegida
- Acceso a ventanas de configuración de localizaciones y sistema desde botones.

Antes de entrar en el detalle de qué método se usó para qué funcionalidad, daremos unas pinceladas sobre el funcionamiento de PyGTK.

El funcionamiento de esta librería se basa en la gestión de eventos.
Al iniciar el programa se crea un objeto de la clase que define la interfaz gráfica.
Las interacciones con la interfaz se producen mediante clicks de ratón o acciones sobre los elementos de la clase.
Estas interacciones hacen llamadas a métodos definidos en la misma clase de la GUI o llaman a métodos y variables de otras clases.

Como opción definitiva, Tkinter

Las limitaciones de PyGTK, consecuencia de ser un port de la versión original, y el hecho de no venir por defecto instalado por defecto en todas las distribuciones de Python, motivaron el cambio de la librería de esta por Tkinter.

Tkinter es de facto la interfaz gráfica de usuario por defecto de Python. (Python Community, 2013)
Esto significa que viene de serie en las distros de Python de los tres sistemas operativos mayoritarios (Windows, Mac OSX y Linux).
El no tener que instalarlo antes de su uso significa un paso menos para el uso del script de cara al usuario final.

Al ser un sistema tan popular, la documentación sobre su uso es de muy fácil acceso. En la misma página wiki de Python sobre Tkinter hay varios manuales que se pueden considerar una referencia para la creación de interfaces de usuario con esta librería.

Aunque el funcionamiento básico de Tkinter es el mismo que el de PyGTK, se basan en la gestión de eventos, las funciones son algo diferentes.
Es por esto que pasaremos a explicar el funcionamiento de cada tipo de widget de esta versión del programa.

Esto sólo pretende ser una explicación del funcionamiento de los widgets, no del programa entero, así que el código fuente de éste se encuentra como un anejo a este proyecto.

La ventana principal

Creación de la ventana principal

Definidas las mismas proporciones que en la interfaz anterior crearemos una ventana con 600 píxeles de largo, que tendrá por tanto, 373 píxeles de ancho.

La creación de la ventana principal solo requerirá de las siguientes líneas:

```
1    root=Tkinter.Tk();
2    root.title('Polux 0.1');
3    root.geometry("600x373+5+5")
4    root.columnconfigure(0, minsize = 200)
5    root.columnconfigure(1, minsize = 200)
6    root.columnconfigure(2, minsize = 200)
7    widget = Ventana_principal()
8    root.mainloop()
```

En la primera línea crearemos un objeto llamado root que será herencia de la clase base Tk.
En la tercera línea definiremos el título de la ventana. La geometría de ésta será definida en la cuarta línea.

Para que el resultado final sea lo más estético posible añadiremos 5 píxeles de margen interior a la ventana del programa. Esto lo haremos con el método ".geometry".
La cadena de texto que le pasaremos al método tendrá que tener la siguiente forma:

```
'wxh±x±y'
```

Siendo "w" el ancho y "h" el alto de la ventana. "x" representa el margen interior horizontal e "y" representa el margen vertical.

Ya que el sistema elegido para la ordenación de los widgets será la creación de filas y columnas, definiremos el ancho de las tres columnas de la ventana desde un principio.
Las líneas de la cuarta a la sexta servirán para crear tres columnas de 200 píxeles de ancho.

En la línea séptima crearemos el objeto widget de la clase Ventana_principal().

La línea octava iniciará el loop de la ventana principal.

Una vez iniciada la ventana principal ésta estará a la espera de eventos por parte del usuario.

Tipografías

Lo primero dentro de la clase Ventana_Principal definir las tipografías a usar en el sistema.
Éstas serán las mismas en toda la ventana así que es mejor definir una sola vez, y al principio, para que así sean más accesibles.
Para la creación de un tipo de letra personalizado recurriremos al módulo "tkFont".
Tendremos que crear un objeto de la clase Font con los atributos deseados.

Ejemplo práctico

A continuación como muestra, la primera fuente creada para esta GUI, que será la tipografía elegida para el encabezado:

```
1  self.Verd15Bold = tkFont.Font(family = 'Verdana', size = '15',
weight = 'bold')
```

En este caso hemos creado una fuente llamada "Verd15Bold" con una tipografía Verdana, un tamaño de 15 y con un estilo "bold" es decir, negrita.

Colocar "self." al principio del nombre nos permitirá acceder a esta fuente desde cualquier parte del programa. (Wouters, 2010).

En caso de no definir ninguna fuente en el script el sistema usará la tipografía por defecto del sistema operativo donde se ejecute. El hecho de tener una cantidad casi ilimitada de combinaciones para nuestras tipografías hace aconsejable el uso de éstas como medio de personalización de nuestro programa.

Parámetros

Algunas de las opciones disponibles son (Shipman, Tkinter 8.5 reference: a GUI for Python, 2013)[119]:

- "family". Familia de la fuente. Varía dependiendo del sistema operativo utilizado.
- "size". Altura de la fuente expresada en puntos con un número entero. Se puede expresar la altura en píxeles, para eso hay que utilizar el prefijo –n.
- "weight". Estilo de la fuente. "bold" para negrita y "normal" para el grueso estándar.
- "slant". Más opciones para el estilo de la fuente. En este caso 'italic' para cursiva y 'roman' para la inclinación normal.
- "underline". Esta opción nos posibilita subrayar las letras. Fijando un '1' activamos esta opción, por defecto estará deshabilitada con un '0'.
- "overstrike". Con este comando podremos tachar el texto. Siguiendo el método habitual con un '1' activaremos esta funcionalidad y con un '0' la desactivaremos.

Métodos públicos

Los objetos de la clase Font también poseerán los siguientes métodos.

.actual()

Este método nos devolverá las características actuales de la clase elegida. Para obtener un valor en concreto tendremos que pasárselo como argumento de la función.

.cget(opcion)

Con este método obtendremos el valor del argumento pasado como opción.

.configure(opcion,…)

Si queremos cambiar una opción (o varias) de un objeto ya creado podremos usar este método para cambiarlas en cualquier parte del código. (Shipman, Tkinter 8.5 reference: a GUI for Python, 2013).

.copy()

Esta función nos devuelve una copia de la fuente.

.measure(texto)

Si pasamos una cadena de texto como argumento a esta función obtenemos el ancho en píxeles que ocupara el texto con esta tipografía.

.metrics(opcion)

Llamando a esta función sin argumentos obtendremos un listado de las características métricas de la tipografía invocada. Podremos llamar, también, a cada característica por separado para así obtener su valor concreto.

---

[119] Páginas 10 a 11.

Ya que las definiciones de las tipografías son diferentes dependiendo del sistema operativo, podremos obtener diferentes resultados con el mismo código.
Las posibles soluciones a este problema las trataremos una vez hayamos explicado todo el script.

Labelframes

La manera más sencilla de agrupar los botones y las etiquetas es mediante la creación de un "labelframe".
Un labelframe, dicho con pocas palabras, es un marco con una etiqueta.
Tanto el marco, como la etiqueta son totalmente configurables.

La sintaxis para crear un labelframe es la siguiente (Shipman, Tkinter 8.5 reference: a GUI for Python, 2013)[120]:

```
w = Tkinter.LabelFrame(padre, opción, ...)
```

Siendo "w" el objeto creado y "LabelFrame" la clase del módulo Tkinter.
Tendremos que definir una clase base (parent) que en nuestro caso será el objeto que define la ventana del programa, root.

Ejemplo práctico

Por poner un ejemplo citaremos el primer labelframe creado en este script.

```
1 ventana_observador = Tkinter.LabelFrame(root, text = 'Observador',
font = self.Verd12Italic, width=190, height=205)
```

Como podemos ver hemos creado el labelframe ventana_observador que pertenece a la clase root, cuya etiqueta es "Observador". La tipografía, creada por el método del punto anterior se llama "self.Verd12Italic" y el tamaño del marco es de 190 píxeles de ancho por 205 píxeles de alto.

Parámetros

Aunque solo hemos definido cuatro parámetros del widget, Tkinter nos permite fijar una amplia variedad de opciones. Las opciones que podemos definir son las siguientes (Shipman, Tkinter 8.5 reference: a GUI for Python, 2013)[121].

[120] Página 50
[121] Página 51

- “bg”. Color de fondo del widget. Si no lo definimos será el del tema predeterminado del sistema operativo.
- “bd”. Ancho de la línea alrededor del widget. Por defecto es de dos píxeles.
- “cursor”. Con esta opción definiremos el tipo de cursor que aparecerá cuando pasemos el ratón sobre el widget.
- “fg”. Este parámetro fija el color de la etiqueta del widget.
- “height”. Dimensión vertical la ventana creada.
- “highlightbackground”. Color de resalte del widget cuando no está seleccionado para la introducción de texto.
- “highlightcolor”. Color de resalte del widget cuando está seleccionado para introducir texto en él.
- “highlightthickness”. Grosor del resalte del widget.
- “labelanchor”. Este parámetro fija la posición de la etiqueta en el marco. Se pueden fijar nueve posiciones diferentes, siendo ‘nw’ noroeste la posición por defecto.
- “labelwidget”. En vez de especificar un texto para la etiqueta del labelframe se puede designar una etiqueta ya creada.
- “padx”. Este parámetro define un margen interior en píxeles en el lado izquierdo y derecho del widget.
- “pady”. Al igual que la opción anterior “pady” también define un margen interior, esta vez el margen será en la parte superior e inferior del widget.
- “relief”. La apariencia de la línea que bordea el widget puede ser modificada con este parámetro.
- “takefocus”. Este widget, al no tener texto u opciones que modificar, normalmente no recibirá ninguna entrada del teclado o ratón. Este comportamiento puede alterarse si definimos este parámetro como cierto, *True*.
- “text”. Texto de la etiqueta de la labelframe. Esta opción será ignorada si se define una label como texto de la ventana.
- “width”. Ancho de la ventana en píxeles.

Como apunte general mencionaremos una peculiaridad del comportamiento del parámetro height y width. Estos sólo serán tenidos en cuenta cuando el flag del método “.grid_propagate()” sea puesto a 0.
Las opciones sobre el posicionamiento de los widgets en las ventanas, al ser las mismas para todos los elementos, serán mencionadas al final de la descripción de los widgets utilizados.

Contexto de utilización en este script

En la ventana principal de este programa hemos utilizado los labelframes para agrupar widgets utilizados para funciones similares.
Hemos creado cinco grupos de objetos agrupados por las siguientes funciones:

- Menú para seleccionar, mostrar y/o modificar un observador.
- Menú para mostrar las coordenadas celestes del satélite elegido.

- Menú para seleccionar un satélite disponible.
- Menú para mostrar y actualizar el tiempo del sistema.
- Menú para configurar diferentes opciones de funcionamiento del programa y para activar las rutinas de seguimiento.

Labels

Una etiqueta es la opción más sencilla para mostrar texto. Posee algunas desventajas en comparación con un widget como "Text" pero, debido a su simplicidad, puede ser la opción más recomendable para mostrar pequeñas cadenas de caracteres.

Las etiquetas pueden mostrar un texto estático o un texto variable. Para mostrar un texto variable tendremos que crear un objeto de la clase StringVar. Vincularemos este objeto a la etiqueta creada.
Entraremos en la definición de los objetos de la clase "StringVar" en el punto siguiente.

Remitiéndonos al mismo manual que en el punto anterior (Shipman, Tkinter 8.5 reference: a GUI for Python, 2013)[122] el método para la creación de una "Label" será el siguiente:

```
1 w = Tkinter.Label(padre, opción = valor, …)
```

"w" será como siempre el widget creado. La clase base será parent, en nuestro caso root. Las diferentes opciones irán junto a la definición de la clase base separadas, como siempre por comas.

Parámetros

Las diferentes opciones para la creación de una label son las siguientes (Shipman, Tkinter 8.5 reference: a GUI for Python, 2013)[123]:

- "activebackground". Color de fondo cuando el cursor del ratón se encuentra sobre el widget.
- "activeforeground". Color de primer plano cuando el cursor del ratón se encuentra sobre la etiqueta.
- "anchor". Este parámetro fija la posición del texto (o imagen) del widget si este posee más espacio del necesario para su contenido.
- "bg". Al igual que en el punto anterior, color de fondo de la etiqueta.
- "bitmap". Si queremos que la etiqueta muestre una imagen de mapa de bits tendremos que fijar esta opción con el nombre de la imagen elegida.
- "bd". Como en el punto anterior, ancho de la línea que bordea a la etiqueta.

[122] Página 48.
[123] Páginas 49 a 50.

- “compound”. Si deseamos mostrar una imagen (del tipo que sea) y texto esta opción nos permitirá seleccionar como se orientará la imagen con respecto al texto. Las opciones disponibles son: “LEFT”, “RIGHT, “CENTER”, “BOTTOM” y “TOP”.
- “cursor”. Esta opción fija el tipo de cursor a mostrar cuando el ratón se encuentra sobre la etiqueta.
- “disabledforeground”. Color de primer plano cuando la etiqueta se encuentra desactivada.
- “font”. Como no podía ser de otra manera con “font” seleccionamos una tipografía creada en esta rutina para personalizar la apariencia del texto.
- “foreground”. Si el widget contiene texto con esta opción podremos fijar el color de este.
- “height”. Este parámetro fija la altura del texto en líneas en vez de en píxeles, como el punto anterior.
- “highlightbackground”. Esta opción, y las dos siguientes, son similares a la gran mayoría de los widgets de Tkinter. Con “highlightbackground” definiremos el color de resalte del widget cuando no tiene activada la entrada por teclado.
- “hightlightcolor”. Color de resalte cuando tiene activada la entrada por teclado.
- “hightlightthickness”. Ancho de la línea de resalte del widget.
- “image”. Para mostrar una imagen que no sea un mapa de bits tendremos que usar esta opción. Crearemos un objeto de la clase Image e igualaremos esta opción con su nombre.
- “justify”. Este parámetro fijará como se orientarán las líneas del texto. Por defecto estarán centradas, CENTER, pero podremos alinearlas a la izquierda, LEFT, o a la derecha, RIGHT.
- “padx”. “padx” nos dará la opción, al igual que en el objeto LabelFrame, de fijar un margen interno a izquierda y derecha del widget.
- “pady”. Del mismo modo “pady” fijará un margen similar esta vez en la parte superior e inferior del objeto.
- “relief”. Este parámetro permite seleccionar el estilo del marco decorativo del widget. Por defecto está fijando en “FLAT”.
- “state”. Al crear el objeto “state” esta opción estará fijada en “NORMAL”. Si por cualquier motivo queremos que la etiqueta se desactive tendremos que fijarla en “DISABLED”. Para recuperar su estado anterior cambiaremos el valor de la opción a “ACTIVE”.
- “takefocus”. Al igual que el widget del punto anterior este objeto por defecto no acepta entradas de teclado. Su funcionamiento también se puede forzar para que si las acepte siempre y cuando se fije este parámetro a *True*.
- “text”. Con “text” podremos fijar el texto de la etiqueta. Este parámetro solo nos permite fijar texto estático.
- “textvariable”. Para fijar un texto variable haremos uso de esta opción. Como hemos mencionado al principio del punto tendremos que igualar este parámetro a la StringVar creada.
- “underline”. “underline” nos permitirá subrayar el texto hasta un determinado carácter del texto. Por defecto está fijado a -1 para así evitar el subrayado de algún elemento.

- "width". Al tratarse de un widget que contiene texto este parámetro definirá el ancho del objeto en caracteres, no en píxeles.
- "wraplenght". Por defecto, si una etiqueta tiene varias líneas de texto estas se interrumpen al usar un salto de línea (\n). Si queremos que estas se corten a una longitud determinada tendremos que determinarla con esta opción.

La gran mayoría de opciones de este elemento no son usadas en este proyecto.
Ya que las etiquetas se usarán para mostrar texto estático y variable las opciones más útiles para este proyecto serán "text" y "textvariable".
"font" y "width" se usarán solo para mejorar la legibilidad del texto.

Ejemplo práctico

Para dar una muestra de las etiquetas más usadas por el sistema mostraremos dos ejemplos.
El primero, la etiqueta para mostrar la cadena de texto "Localizacion".

```
1 etiqueta_localizacion = Tkinter.Label(ventana_observador, text = 'Localizacion', font = self.Verd12Roman)
```

Podemos comentar varios puntos sobre esta definición:

- Lo primero es la ausencia del prefijo "self." en la definición del nombre del objeto. Éste no será necesario al no necesitar modificar este objeto fuera de su función. Las etiquetas con texto fijo no son modificadas en este script.

- La clase base del objeto también será digna de mención. En este caso ya no será "root", sino "ventana_observador". Queremos que la etiqueta pertenezca al marco que agrupa las rutinas relacionadas con la posición del observador.

- La definición de la cadena de texto no reviste de ninguna complicación. Como detalle curioso nos fijaremos en que la definición de la fuente sí que tiene el prefijo antes mencionado. El haber definido todas las tipografías en una función aparte nos fuerza a ello.

El segundo, la etiqueta para mostrar el observador actual del sistema.

```
1 mostrar_localizacion = Tkinter.Label(ventana_observador, textvariable = self.text_var_localizacionbuena, font = self.Verd12Roman)
```

El único punto remarcable con respecto al anterior punto es la definición del texto de la etiqueta. En este caso hemos igualado la opción "textvariable" al objeto "self.text_var_localizacionbuena".

En este script los objetos del tipo StringVar se suelen definir con el prefijo “self.”.
Esto es debido a que su valor se fija en una función diferente a la función donde se crean.

StringVars

En el capítulo anterior hemos mencionado los objetos del tipo StringVar pero no hemos entrado en su definición.
Trataremos ahora de dar las pautas de su funcionamiento y como hemos de crearlos correctamente.

Los objetos de tipo StringVar pertenecen a la misma categoría que los objetos de tipo IntVar, DoubleVar y BoleanVar (Lutz, 2011)[124].
Aunque estos objetos definan valores no son variables al uso. Son la manera que tiene un objeto de Tkinter de acceder a un valor que puede cambiar, o no, durante la ejecución del script.

Recurriremos otra vez al manual de referencia de J. Shipman para buscar la forma correcta de crear un objeto de este tipo (Shipman, Tkinter 8.5 reference: a GUI for Python, 2013)[125].

```
1 v = Tkinter.StringVar()
```

Una StringVar se puede definir sin tener que referenciarla a un objeto base.
El texto de la StringVar se definirá posteriormente usando los métodos del objeto.

Métodos públicos de StringVar()

Este objeto solo tiene dos métodos para trabajar con él:

.get()

Con este método obtendremos el valor actual de la StringVar seleccionada.

.set(valor)

Este método nos permitirá fijar el valor de la StringVar elegida. “valor” puede ser cualquier cadena de caracteres válida. Incluso una cadena vacia del tipo ‘’.

Ejemplo práctico

Como en los objetos anteriores pondremos un ejemplo de funcionamiento de este objeto. Añadiremos a nuestra explicación, para hacerla más comprensible, un uso práctico de una StringVar en un objeto del tipo Label.

---

[124] Página 454.
[125] Página 154.

```
1 self.text_var_localizacionbuena = Tkinter.StringVar()
2 self.text_var_localizacionbuena.set('')
3     mostrar_localizacion     =     Tkinter.Label(ventana_observador,
textvariable     =     self.text_var_localizacionbuena,     font     =
self.Verd12Roman)
.
.
.
1 self.text_var_localizacionbuena.set(observador[0])
```

En la línea 1 crearemos la StringVar con nombre "self.text_var_localizacionbuena".
El valor del objeto se dará en la línea 2. Como no queremos mostrar nada por pantalla en un principio le daremos un valor vacio ' '.
La etiqueta "mostrar_localizacion" hará uso de esta StringVar para mostrar el texto, de momento, vacío.

No será hasta otro momento cuando se fije el valor del objeto. En este caso será el ítem primero de la lista observador.

Buttons

Un botón es quizás el elemento más fácilmente reconocible de una interfaz gráfica.
Es la opción más sencilla para hacer una llamada a una función.

Aunque los botones se suelen usar solo para llamar a una rutina, también disponen de un amplio abanico de opciones de personalización. El tamaño y el diseño del botón es modificable, además de la apariencia del texto que contiene.

Como todos los widgets, la creación de un botón se lleva a cabo mediante una clase del módulo Tkinter (Shipman, Tkinter 8.5 reference: a GUI for Python, 2013)[126]

```
1 w = Tkinter.Button(padre, opcion = valor, …)
```

Tendremos que definir la clase padre a la que pertenece y el valor de las opciones que queramos modificar.

Parámetros

Algunas de las opciones de las que disponemos son similares a los objetos anteriormente citados. Otras son exclusivas de este widget.
Pasaremos a citarlas todas a continuación (Shipman, Tkinter 8.5 reference: a GUI for Python, 2013)[127]:

- "activebackground". Color de fondo del botón cuando el cursor del ratón está sobre él.

[126] Página 18.
[127] Página 18 a 20.

- “activeforeground”. Color de primer plano cuando el ratón se encuentra sobre el widget.
- “anchor”. Posición del texto dentro del botón. Se puede colocar en las mismas nueve posiciones de siempre.
- “bd”. Ancho de la línea que bordea el botón. Por defecto es de dos píxeles.
- “bg”. Color de fondo del objeto.
- “bitmap”. Nombre de una de las imágenes de mapa de bits estándares.
- “command”. Igualando esta opción a una función o método al hacer click en el botón este la llamará.
- “cursor”. Opción para determinar qué tipo de cursor habrá encima del widget. Por defecto será el perteneciente al tema del sistema operativo en uso.
- “default” Está opción será interesante si queremos que el botón esté inoperativo por un determinado motivo. Fijándola a “NORMAL” el botón estará operativo pero si la fijamos a “DISABLED” el botón dejará de funcionar.
- “disabledforeground”. Color de primer plano del botón cuando “default” está igualado a “DISABLED”.
- “fg”. Color de primer plano del texto en condiciones normales del objeto.
- “font”. Fuente del objeto. Como en el resto del objeto igualaremos esta opción a nuestra tipografía personalizada.
- “height”. Si el botón contiene texto está opción fija el número de líneas de altura que tendrá este. Si es una imagen de mapa de bits con este parámetro fijaremos la altura en bits del widget.
- “highlightbackground”. Color de resalte del botón cuando no está seleccionado para la entrada de teclado.
- “highlightcolor”. Color de resalte del widget cuando si está seleccionado.
- “highlightthickness”. Ancho de la línea de resalte de la selección para la entrada de teclado en la rutina.
- “image”. Como en el widget Label, los botones también permiten usar objetos del tipo Image. Con este parámetro seleccionaremos la imagen deseada.
- “justify”. Como justificaremos el texto en el supuesto de tener varias líneas de éste. Los métodos de justificación son los usuales. Centrada, a la izquierda y la derecha.
- “overrelief”. Este parámetro nos da la opción de fijar el estilo del botón cuando el cursor del ratón pasa por encima del botón.
- “padx”. Espacio dentro del widget entre el texto y el lado izquierdo y derecho del botón.
- “pady”. Margen entre el texto y la parte superior e inferior del botón.
- “relief”. Opción para fijar el estilo del botón cuando no está el cursor del ratón encima de él. Por defecto es “RAISED”.
- “repeatdelay”. Los botones normalmente al pulsarlos se iluminan solo una vez. Si queremos que se iluminen varias veces tendremos que fijar este valor y el siguiente, “repeatinterval”. El valor (en milisegundos) que le asignemos a este parámetro será el tiempo mínimo que el usuario tendrá que pulsar el botón para que este empiece a parpadear. Cuando deje de pulsar el botón este dejará de parpadear.

- "repeatinterval". Este parámetro fijará la duración de los parpadeos del botón. Como la opción anterior el valor se dará en milisegundos.
- "state". Con esta opción podremos desactivar el botón para así evitar pulsaciones indeseadas.
- "takefocus". Los botones son widgets que por defecto vienen preparados para recibir entradas desde el teclado. Pulsando la tecla espaciadora podemos clickar un botón como si fuera el ratón. Si queremos desactivar este comportamiento para evitar problemas asignaremos el valor '0' a este parámetro.
- "text". Con este parámetro asignamos la cadena de caracteres que queremos que muestre el botón por pantalla.
- "textvariable". Al igual que en las etiquetas también podemos usar StringsVar y objetos de ese tipo en los botones. Con esta opción las asignamos.
- "underline". Con "underline" podemos subrayar caracteres de la cadena de texto del botón. Por defecto este valor está fijado a -1. El número que fijemos será la cantidad de caracteres que se subrayen.
- "width". Parámetro para fijar el ancho del botón. En caracteres si estamos usando texto o en bits si estamos mostrando imágenes.
- "wraplenght". Esta opción nos da la posibilidad de cortar las líneas de texto del widget a una longitud determinada. Tendremos que asignarle un valor en caracteres.

Ejemplo práctico

Un ejemplo de la creación de un botón puede ser el siguiente.

```
1 self.boton_configuracion = Tkinter.Button(ventana_controles, text =
'Configuracion', font = self.Verd12Roman, height = 1, width = ancho,
command = self.configurar_sistema)
```

Como podemos ver la mayoría de las opciones definidas en este botón son las usuales.
Las únicas diferencias estriban en el valor de "width" y en el parámetro command.

Un punto que aún no hemos mencionado, es cómo vamos a lograr que el aspecto del sistema sea el mismo independientemente del sistema operativo que utilicemos.
Para ello definiremos los tamaños de algunos objetos con variables. Estas variables serán distintas dependiendo del sistema operativo que usemos.
Más adelante explicaremos el funcionamiento de esta rutina.

Volviendo a la rutina de creación del botón, una vez invocada nos dará por pantalla el siguiente objeto.

Figura 41. Botón de ejemplo "Configuración" en Mac OS X (10.8.5).

Este es el aspecto que nos daría el botón corriendo el script en Mac OS X. Como hemos mencionado, el aspecto diferirá dependiendo del sistema operativo desde el que ejecutemos el código de nuestro programa.

El mismo botón en un entorno X11 (sistema operativo Raspbian), tendrá un aspecto algo distinto.

Figura 42. Botón de ejemplo "Configuración" en Raspbian.

En los sistemas operativos Windows el aspecto también variará algo.

Figura 43. Botón de ejemplo "Configuración" en Windows XP.

En los dos puntos finales de esta sección, explicaremos como modificar el código para que la apariencia de los elementos sea lo más parecida posible.

Notebook

Este objeto es similar a una libreta. Su utilidad se asemeja al de un widget de la clase Frame o Labelframe, siendo su uso más común el de contenedor de otros elementos.
La diferencia con los anteriormente mencionados, estriba en que un objeto de la clase Notebook puede contener diferentes elementos en cada uno de sus paneles.

La creación de un Notebook se realiza mediante la creación de un objeto del módulo ttk.

El módulo ttk es una actualización del módulo Tkinter. Además de agregar algunos nuevos widgets que no estaban disponibles en Tkinter este módulo añade la posibilidad de crear temas.

Un tema es un conjunto de opciones predefinidas para el diseño gráfico de una interfaz. La gran ventaja de esto es que facilita enormemente el diseño de una interfaz que corra en múltiples plataformas.

No entraremos en la explicación de las ventajas que proporciona ttk.
Para el propósito inicial de este proyecto con el uso del módulo Tkinter basta. De todas maneras, se estudiará la posibilidad del uso de ttk para mejorar la legibilidad de la interfaz en las diversas plataformas.
Este tema será abordado en el último capítulo de esta memoria.

El proceso de creación de un widget del tipo Notebook seguirá el mismo modelo que los widgets anteriores.

```
w = ttk.Notebook(padre, opcion = valor, …)
```

La única diferencia estriba en que, esta vez, el módulo será ttk en vez de Tkinter.

Como es habitual, a continuación, detallaremos las diferentes opciones sobre el objeto que nos brinda ttk al crearlo (Shipman, Tkinter 8.5 reference: a GUI for Python, 2013)[128].

Parámetros

- "class_". En el supuesto de que queramos que el objeto herede sus características de una clase diferente a la clase Notebook tendremos que especificarlo aquí. Hay que tener en cuenta que una vez definido esto ya no se puede modificar, a no ser que se destruya el objeto.
- "cursor". Nos permite seleccionar que modelo de cursor se verá al pasar este sobre el Notebook.
- "height". Este parámetro, al igual que en muchos otros objetos, fija la altura del objeto.
- "padding". Opción similar a padx y pady. Con este parámetro podemos definir el espacio extra que tendrá el widget alrededor suya.
- "style". Parámetro propio del módulo ttk que nos permite definir el estilo del objeto.
- "takefocus". Por defecto un Notebook está preparado para recibir entradas desde el teclado. Mediante estas entradas podremos seleccionar las diferentes pestañas del widget. Si quisiéramos desactivar esta funcionalidad tendríamos que fijar este parámetro a un valor falso. Por ejemplo `takefocus = False`.
- "width". Del modo usual, con esta opción fijamos el ancho del Notebook.

Métodos públicos

Ya que tenemos que interactuar con las páginas del objeto, los métodos de este widget serán más numerosos que los habituales.
Entre ellos podemos encontrar los siguientes (Shipman, Tkinter 8.5 reference: a GUI for Python, 2013)[129]

.add(hijo, **kw)

Con este método añadimos un widget heredado "hijo" a un nuevo panel del Notebook. Este nuevo panel se colocará por defecto en la última posición.

---

[128] Página 126 a 127.
[129] Página 127 a 128.

Si el objeto heredado child ya existe, y está oculto, con esta rutina volverá a aparecer en pantalla.

Las opciones disponibles se mencionan en el siguiente punto.

.enable_transversal()

Llamando a este método obtendremos algunas funcionalidades extra para las entradas de teclado.

- Mediante la combinación de teclas *Control+Tab* recorreremos los paneles del Notebook. Cada vez que la pulsemos avanzaremos un panel hacia delante. Si nos encontramos en el último panel del widget al pulsar estas dos teclas iremos al primer panel del objeto.
- La pulsación de las teclas *Mayúsculas+Control+Tab* recorrerá el trayecto inverso. Al igual que la combinación anterior ésta tendrá la peculiaridad de que saltará del primer panel al último.
- La última opción que nos proporciona este método es el salto a un panel en particular mediante la combinación del teclado *Alt+X.* Para hacer uso de esta funcionalidad tendremos que subrayar una letra de cada etiqueta de todos los paneles. Substituyendo X por la letra subrayada podremos acceder a esta pestaña en concreto.

Hay que tener en cuenta que estas características solo estarán disponibles si no se ha deshabilitado la entrada de teclado para este objeto.

.forget(hijo)

Llamando a este método con el nombre del widget hijo de nuestro objeto lo eliminamos permanentemente del Notebook.

.hide(hijo)

A diferencia del punto anterior, con este método no eliminamos el widget sino que lo escondemos. Como mencionamos anteriormente llamando a la función .add podremos recuperar el objeto en el Notebook.

.index(tabId)

Invocando este método con el nombre de un panel como argumento obtenemos su indice. Esta función tiene una peculiaridad, y es que si la llamamos con la string “end” nos devuelve el número de paneles.

.insert(donde, hijo, **kw)

Con esta función introducimos el widget “hijo” en un nuevo panel a continuación del panel “donde”. También podemos fijar, dentro de la misma función, los parámetros del nuevo panel.

.select([tabId])

Esta rutina podrá actuar de dos formas distintas.

- Si la invocamos sin ningún argumento nos devolverá el nombre del panel donde se encuentra el Notebook.
- Si llamamos a esta función dándole como argumento el nombre de un panel nos mostrará este por pantalla.

.tab(tabId, opcion=None, **kw)

Con este método podremos modificar las opciones de un panel una vez creado. Para ello tendremos que llamar a esta rutina dándole como argumentos el nombre del panel y los parámetros a definir.

Estos parámetros se detallan en el siguiente punto.

.tabs()

Invocando este método obtenemos una lista ordenada (del primero al último) de los paneles del Notebook.

Opciones para el método público .add() y .tab()

Estos dos métodos tienen una serie de opciones que podremos configurar para variar la apariencia de nuestro objeto. Ya que ambos métodos comparten los mismo parámetros los mencionaremos en el mismo punto (Shipman, Tkinter 8.5 reference: a GUI for Python, 2013)[130].

- "compound". Este parámetro solo tendrá utilidad en el caso de que añadamos a la pestaña de nuestro panel texto e imágenes a la vez. Mediante esta opción configuramos la posición de la imagen que agregemos con respecto al texto. Tendremos cinco posiciones posibles: BOTTOM, TOP, LEFT, RIGHT y CENTER.
- "padding". Con esta opción añadiremos un margen entre el contenido agregado y las paredes internas del panel.
- "sticky". Esta opción nos permitirá definir la posición del objeto agregado en el caso de que disponga de más espacio del necesario. Las posiciones disponibles son las mismas que las definidas para el método grid(). Estas están definidas en la figura 4.7 de este mismo capítulo.
- "image". Con este parámetro añadimos una imagen a la pestaña del panel que estemos configurando.
- "text". Igualando "text" a una cadena de caracteres podremos escribir texto en la pestaña del panel.
- "underline". El función de "underline" se asemeja al de widgets anteriores. El texto de la pestaña del panel tendrá tantos caracteres subrayados como valor tenga este parámetro. Por defecto "underline" vale cero.

Ejemplo práctico

El código fuente para crear el objeto será el siguiente:

[130] Página 128.

```
1 estados_notebook = ttk.Notebook(ventana_controles, height = 80,
width = 205)
2 primera_ventana = ttk.Frame(estados_notebook)
3 estados_notebook.add(primera_ventana, text='Conexion')

4 segunda_ventana = ttk.Frame(estados_notebook)
5 estados_notebook.add(segunda_ventana, text='2')

6 tercera_ventana = ttk.Frame(estados_notebook)
7 estados_notebook.add(tercera_ventana, text='3')
```

Detallaremos a continuación la función de cada línea del código anterior.

En la primera línea hemos creado el objeto Notebook. El widget padre será ventana_controles. Solo hemos definido dos parámetros, altura (height) y ancho(width).

En la segunda, cuarta y sexta línea hemos creados objetos del tipo Frame.
Estos pertenecen al módulo ttk para evitar posibles incompatibilidades con el objeto Notebook.

Secuencialmente en las líneas tercera, quinta y séptima hemos añadido tres paneles al objeto. En estos tres paneles hemos definido el valor de su pestaña y hemos agregado los tres widgets del tipo frame creados anteriormente.

El valor del texto es, respectivamente, “Conexión”, “2” y “3”.

Al no haber definido el estilo del widget Python optará por el estilo predefino.
Esto nos dará la siguiente captura de pantalla en Mac OS X.

Figura 44. Ejemplo de notebook.

Aunque esté objeto se encuentra vacio de contenido, no tiene ningún elemento en sus paneles para interactuar con él, en los capítulos siguientes discutiremos que utilidad otorgarle y con que elementos darsela.

Scrollbars

El último objeto que nos falta por detallar son las Scrollbars.
He dejado este para el último porque es el único que no es nativo de la librería Tkinter.

Esta librería adolece de un defecto y es que no dispone de ninguna lista con controles deslizantes. Esta ausencia se puede subsanar creando una clase heredada con los atributos necesarios.

Hemos recurrido a la clase creada por John W. Shipman llamada "ScrolledList".
La documentación sobre esta clase es de dominio público (Shipman, ScrolledList: A Tkinter scrollable list widget, 2009)[131].

Según su documentación, el método para crear la clase será el siguiente (Shipman, ScrolledList: A Tkinter scrollable list widget, 2009)[132].

```
1 s = ScrolledList (padre, width = W, height = H, vscroll = VS,
hscroll = HS, callback = c)
```

"s" será como siempre el objeto creado.
"padre" será la clase base donde creemos el objeto. En nuestro caso será una de las cinco labelframe de la ventana principal.
"width" estará expresado en caracteres y será el ancho de nuestra widget.
La altura de la ventana estará en líneas. Se definirá fijando el valor de H.
Por defecto, la clase creará un control deslizante vertical. Si por algún motivo no lo deseas tendremos que igualar vscroll a 0.
El constructor de la clase no creará un control deslizante horizontal. Para crearlo tendremos que igualar hscroll a 1.
Cuando pulsemos en algún elemento de la lista se producirá una llamada a una función. Esta se definirá, al igual que los demas objetos, igualando el nombre de la función al parámetro callback.

Las funciones a las que llamemos tendrán que tener una peculiaridad. Tendremos que pasarles en su definición el valor de línea seleccionada.

Ejemplo práctico

Para mostrar esto pondremos un ejemplo de la creación de una ScrolledList y de la función a la que llama.

La creación del objeto se hará del siguiente modo:

```
1 self.lista_ciudades = scrolledlist.ScrolledList
(ventana_observador, width=10, height=2, hscroll=1,
callback=self.seleccionar_ciudad)
```

No entraremos en la definición de las opciones ya que están explicadas en los parrafos anteriores.

---

[131] http://infohost.nmt.edu/tcc/help/lang/python/examples/scrolledlist/scrolledlist.pdf
[132] Página 3.

La definición de la función a la que invocamos tendrá que tener obligatoriamente un elemento que será el número de línea seleccionada.
Podemos dar a esta variable el nombre que queramos. Por ser más descriptivos en la función la hemos llamado "indice_ciudad".

```
1 def seleccionar_ciudad(self, indice_ciudad):
• …
• …
• …
```

Si se nos olvidara nombrar una variable para definir la línea del programa el interprete de Python nos avisaría del error.

Métodos públicos de la clase ScrolledList

Esta clase también tiene definidos algunos métodos públicos para interactuar con ella (Shipman, ScrolledList: A Tkinter scrollable list widget, 2009)[133].

.count()

Devuelve el numero de líneas del widget. Se tiene que invocar sin ningún argumento.

.__getitem__(self, i)

Devuelve el elemento de la línea "i".

.append(s)

Añade la cadena de caracteres "s" como una nueva línea en el widget. Es similar al método de mismo nombre de las listas.

.insert(linex, s)

Inserta la cadena de caracteres "s" una posición antes de la línea numero "linex".

.delete(linex)

Elimina la línea "linex" de la lista.

.clear()

Este método elimina todas las líneas del widget.

Métodos de alineación

Tkinter posee varios métodos para ordenar los elementos en la interfaz gráfica.
La conveniencia de usar uno u otro depende del programador, ya que, ambos comparten similitudes y, de cara al usuario, no presentan diferencias.

---

[133] Página 4.

Aunque se pueden mezclar dentro del mismo código esta es una práctica no recomendable. Si aún así fuera necesario hacerlo, la forma más correcta de hacerlo sería separando los elementos alineados con cada método en diferentes marcos. En el libro de referencia Programming Python de Lutz (Lutz, 2011) [134] podemos encontrar varios ejemplos de combinaciones mixtas de métodos en el mismo script.

Haremos una introducción somera al metodo "Pack()" y al metodo "Place()" para luego explicar con más detalle el método "Grid()", sistema utilizado en este programa.

Pack()

El método "Pack()" se usa para comunicarse con "Packer", un gestor de diseños que se encarga de los objetos de las clases heredadas colocandolos en orden en los objetos de las clases padre (Grayson, 2000)[135].

Este método no recibe herencia de ningún otro. Mediante la configuración de sus diferentes opciones fijamos la posición de cada objeto con respecto a los demas.
La gran desventaja de este método es que no permite fijar posiciones absolutas.

Parámetros

Las opciones para establecer el diseño son las siguientes (Grayson, 2000)[136]:

- "after" y "before". Aunque estas dos opciones aparezcan en este libro han sido sustituidas por "anchor" (Python Software Foundation, 2013)[137].
- "anchor". Opción para fijar la posición del widget. Por defecto es "CENTER".
- "expand". Fijando este parámetro podremos activar o desactivar que el widget se expanda hasta ocupar todo el espacio posible. Los valores de esta opcion tendrán que ser boleanos.
- "fill". Con esta opción, si "expand" es igual a 1, fijamos como se expandirá el widget por el espacio sobrante. Las opciones serán:
    - NONE: el widget no se expande, opción por defecto.
    - X: se expande horizontalmente.
    - Y: se expande verticalmente.
    - BOTH: la expansión es horizontal y verticalmente.
- "in_". Si queremos que el widget se encuentre dentro de otro widget (no de una clase base) tendremos que ajustarlo con este parámetro. No es recomendable utilizar esta opción.

---

[134] Páginas de 568 a 570.
[135] Página 511.
[136] Páginas de 511 a 512.
[137] http://docs.python.org/2/library/tkinter.html#packer-options

- "ipadx". Margen interior del widget en los laterales izquierdo y derecho. Esto especifica el espacio mínimo vacio que habrá. Por defecto será 0.
- "ipady". Al igual que el parámetro anterior, "ipady" especifica el espacio mínimo vacio. Esta vez será entre sus extremos superior e inferior y el contenido del widget.
- "padx". Con esta opción especificaremos cuanto espacio queremos dejar entre el widget y los laterales izquierdo y derecho del objeto que lo contiene.
- "pady". El parámetro "pady" tendrá una función similar a la anterior, solo que este fijara el espacio entre el widget y sus extremos superior e inferior.
- "side". Especifica a que lado del objeto contenedor se va a empaquetar el widget. Si no se fija un valor diferente por defecto es "TOP".

Métodos públicos de pack()

Este método de alineación también pone a disposición del programador diferentes métodos públicos para modificar, aún más, diversos parámetros.

Los métodos de "pack()" serán:

.pack(opcion = valor, …)

Este método será el necesario para empaquetar un objeto. Con él definiremos los diferentes valores, generalmente relacionados con como empaquetar el widget.

.pack_configure(opcion = valor, …)

Una vez creado el objeto con esta rutina podremos variar sus parámetros.

.pack_forget()

Remueve el widget pero no lo destruye. Invocando de nuevo "pack()" podremos traerlo de vuelta a la ventana.

.pack_info()

Si llamamos a esta función sin argumentos nos devuelve los valores de las opciones del objeto seleccionado.

.pack_propagate(flag)

El widget contenedor se expande tanto como sea necesario para contener los widgets herederos. Por defecto activada, si queremos que no lo esté tendremos que fijar el flag a 0.

.pack_slaves()

Devuelve una lista de los objetos que tiene empaquetado el widget elegido.

Ejemplo práctico

Un ejemplo de un empaquetado mediante el método "pack()" puede ser el siguiente.

Figura 45. Ejemplo de empaquetado con el método pack.

El código fuente del ejemplo anterior será el siguiente:

```
1  import Tkinter

2  root = Tkinter.Tk()
3  root.title("pack()")
4  root.geometry("180x100")

5     etiqueta1   =   Tkinter.Label(root,   text="Primera",   bg="red",
fg="white")
6  etiqueta1.pack(fill=Tkinter.BOTH)
7     etiqueta2   =   Tkinter.Label(root,   text="Segunda",   bg="green",
fg="white")
8  etiqueta2.pack(fill = Tkinter.BOTH, padx = 10)
9     etiqueta3   =   Tkinter.Label(root,   text="Tercera",   bg="blue",
fg="white")
10  etiqueta3.pack(fill=Tkinter.BOTH)
11  etiqueta4   =   Tkinter.Label(root,   text   ="Cuarta",   bg="yellow",
fg="white")
12 etiqueta4.pack(fill=Tkinter.BOTH, pady = 10)

13 root.mainloop()
```

En la primera línea importamos el módulo para crear la interfaz.
Las líneas dos, tres y cuatro corresponden a la creación de la ventana.
La línea trece inicia la ventana.

En la ventana hay tres etiquetas colocadas una después de otra. Los objetos están nombrados por el orden de colocación que tienen.
En este ejemplo podremos comprobar la utilidad del comando "padx" y "pady".
La etiqueta 2 tiene un margen interior de 10 píxeles. En la imagen vemos que hay una area en blanco a los dos lados de la etiqueta de diez píxeles en total
La etiqueta 4 también tiene un margen de 10 píxeles sin embargo, en este caso, en su extremo superior e inferior.

Este código es un ejemplo muy simple, pero muy claro, de la filosofía de este método de empaquetamiento. Amontonar los widgets de una determinada manera.
Aunque muy fácil y rápido de implementar como podemos ver este método no nos permitirá diseños más elaborados.
Para ello tendremos que acudir a uno de los dos sistemas siguientes.

Place()

En este método las posiciones de los objetos están fijadas por medio de coordenadas.
Estas coordenadas pueden estar referidas a la ventana en la que se encuentra el objeto o a otro objeto del mismo script. Este método también nos permite fijar el tamaño en pantalla de los widgets.

Parametros

Este método es poco usado y es dificil dar con documentación sobre él, pero algunos manuales online incluyen información sobre su uso (tutorialspoint)[138].

Las diferentes opciones que podremos configurar al colocar el widget son:

- "anchor". Posición del widget dentro del elemento base. El valor por defecto de este parámetro es "NW".
- "bordermode". Con este parámetro definidos como fijar la posición del objeto. Por defecto el objeto estará dentro del objeto, así que este parámetro valdrá "INSIDE". También se puede fijar la posición fuera del objeto con "OUTSIDE".
- "height". Altura del widget en píxeles.
- "width". Ancho del widget en píxeles.
- "relheight". Este parámetro y el siguiente sirven para fijar al altura y el ancho del widget en comparación con la objeto base donde se coloca. Este valor (en número flotante) variará entre 0.0, el objeto no es visible, hasta 1.0, el objeto ocupa todo el objeto base. Esta opción se encargará del alto del objeto.
- "relwidth". Y esta otra del ancho.
- "relx". Posición relativa del objeto con respecto al objeto base. Esta posición está dada como un objeto flotante entre 0.0 y 1.0. Este parameto fija la posición horizontal. Su valor por defecto es 0.0
- "rely". Con este parámetro fijamos la posición vertical. Su valor por defecto es 0.0.
- "x". Posición absoluta del widget con respecto al objeto base. Se da como un valor entero. Esta opción define la posición horizontal en píxeles. El valor por defecto es 0.
- "y". Posición absoluta vertical. También definida como un numero entero en píxeles. Su valor por defecto también es 0.

[138] http://www.tutorialspoint.com/python/tk_place.htm

Ejemplo práctico

Para entender el funcionamiento de este objeto crearemos una ventana con un widget base y varios puestos alrededor suya.

El código fuente del script será el siguiente.

```
1  import Tkinter

2  root = Tkinter.Tk()
3  root.title("place()")
4  root.geometry("300x300")

5   izquierda_arriba = Tkinter.Button(root, width = 10, height = 2,
wraplength = 10, text ="Izquierda arriba")
6  izquierda_arriba.place(relx = 0.0, rely = 0.0)

7  centro = Tkinter.Button(root, width = 10, height = 1, wraplength =
10, text = "centro")
8 centro.place(relx = 0.5, rely = 0.5)

9   centro_abajo = Tkinter.Button(root, width = 10, height = 2,
wraplength = 10, text = "Centro abajo")
10 centro_abajo.place(x = 150, y = 200)

11  derecha_abajo = Tkinter.Button(root, width = 10, height = 2,
wraplength = 10, text = "Derecha abajo")
12 derecha_abajo.place(anchor = Tkinter.SE, relx = 1.0, rely = 1.0)

13 root.mainloop()
```

El cual nos dará la siguiente ventana.

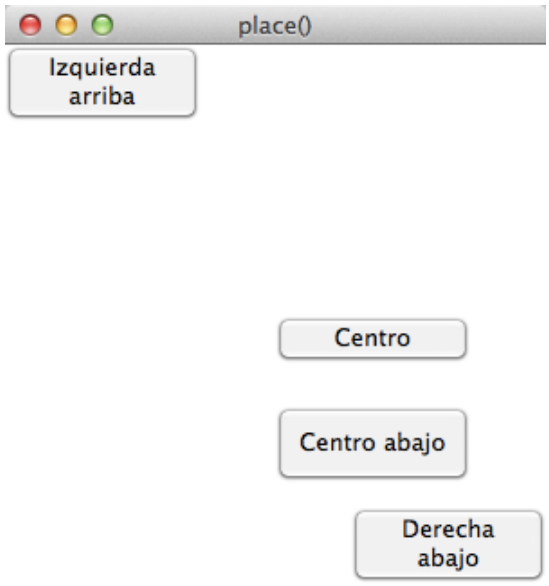


Figura 46. Ejemplo de empaquetado con place.

Con la captura de pantalla de la ventana del script en mente explicaremos el código fuente del programa.

Las líneas dos, tres y cuatro corresponden a la creación de la ventana.
Ventana que se iniciará en la última línea del script, la número trece.

El primer botón, izquierda_arriba, tendrá el texto "Izquierda arriba".
Estará empaquetado haciendo uno de "relx" y "rely". Como queremos que esté lo más pegado a la esquina superior izquierda, el valor de ambos será 0.0.
Esto es posible porque el valor de "anchor" será el predeterminado, el cual es NW.

El segundo botón, centro, con el texto "Centro" estará definido en las líneas de siete y ocho.
También empaquetado con los parámetros "relx" y "rely" cuyo valor será 0.5. Con el valor por defecto de "anchor", la esquina superior derecha del botón se encontrará en el centro de la ventana root.

El tercer botón, centro_abajo, tendrá un modo de empaquetado distinto.
Con el texto "Centro abajo" su empaquetado se hará por medio de valores absolutos.
La esquina superior derecha del boton se encontrará a 150 píxeles en el eje x y 200 píxeles en el eje y de la esquina superior derecha de la ventana root.
Esta definición se encuentra en las líneas nueve y diez.

El cuarto y último botón de la ventana derecha_abajo tiene un elemento diferente al resto de botones. Este es el parámetro "anchor".
Al fijar un valor SE conseguimos que se pueda colocar el elemento atendiendo a su esquina inferior derecha. Así, fijando un valor de relx y rely de 1.0 el botón quedará en la posición de la captura de pantalla.

El principal inconveniente de este método radica en la dificultad de crear ventanas redimensionables. Corremos el riesgo de que, al variar el sistema operativo o el tamaño de la ventana, la apariencia de estas sea muy distinta a la deseada.

Grid()

Es el método usado en este script.
Los elementos se organizan por medio de tablas. Un mismo objeto puede ocupar varias filas o columnas a gusto del programador.
Las filas y columnas se pueden crear dentro de una ventana o de un widget del tipo Frame o LabelFrame.
Este método combina los mejores aspectos de los métodos anteriores. Permite empaquetar elementos dentro de otros elementos sin tener que fijar posiciones pero también permite, si así lo deseamos, fijar posiciones relativas atendiendo las distancias entre los objetos.

Parámetros

Siguiendo la misma tónica que en los puntos anteriores, empezaremos explicando las diferentes opciones para la configuración de este método.

Los parámetros que podemos ajustar son los siguientes (Shipman, Tkinter 8.5 reference: a GUI for Python, 2013)[139]:

- "column". Número de columna desde la que parte el widget. El valor por defecto es 0.
- "columspan". Número de columnas que ocupará el widget. Aunque el valor por defecto es 1 un objeto puede ocupar tantas columnas como quiera.
- "in_". Este parámetro fija el widget que vayamos a colocar como heredado de otro widget.
- "ipadx". Margen interno del widget. Al igual que en puntos anteriores, este parámetro define el espacio mínimo libre que habrá dentro del widget en sus laterales izquierdo y derecho.
- "ipady". Con la misma filosofía del punto anterior, esta opción define el espacio libre entre el contenido del widget y los extremos superior e inferior de este.
- "padx". Espacio mínimo libre entre el widget heredado (el que estamos colocando) y el widget base. Ese valor está referido a sus extremos izquierdo y derecho.
- "pady". Espacio mínimo libre entre el objeto a colocar y el widget donde se coloca, esta vez, atendiendo a su extremo superior e inferior.
- "row". Fila en la que queremos que esté nuestro widget. El número por defecto será el número de la primera fila libre del objeto.
- "rowspan". Número de filas que queremos que ocupe nuestro objeto. Al igual que con la opción "columnspan", este parámetro podrá tomar el valor que deseemos.
- "sticky". Con esta opción fijamos donde se colocara el widget si este tiene espacio de sobra en su localización.

Métodos públicos de Grid()

Grid posee diferentes métodos públicos para interactuar con él una vez colocado el objeto.

.grid_bbox(column, row, col2, row2)

Este método nos dará diferentes salidas dependiendo de los parámetros que le pasemos. Los tres posibles comportamientos serán los siguientes:

- Si no le pasamos ningún argumento nos devolverá una tupla de cuatro valores. Los dos primeros serán la columna y la fila de la esquina superior del wigdet. Los dos ultimos (col2 y row2) serán el tamaño en columnas y filas que ocupa nuestro elemento.

[139] Página 6.

- Si le pasamos el valor de column y row la rutina nos devolverá el valor de la celda que se encuentra en esa columna y fila.
- Si por el contrario le damos el valor de col2 y row2 este método nos devolverá el valor del area total. Area definidida por el widget que empieza en column y row.

.grid_forget()

Con este método podemos hacer que el widget deje de ser visible. Podemos hacer que vuelva a la pantalla usando .grid() pero no recordará su posición.

.grid_info()

Invocando esta función sin argumentos nos devolverá los valores de las diferentes opciones de empaquetado del widget.

.grid_location(x, y)

Dando a esta rutina un par de valores de posición del widget nos devolverá a que celda corresponde esa ubicación.

.grid_propagate()

Por defecto todos los widgets empaquetados con este método ocupan todo el espacio posible. Cualquier asignación de un tamaño diferente al estandar es ignorado. Si por razones de diseño, queremos que solo ocupen el tamaño fijado en su creación tendremos que invocar esta rutina con un argumento falso (0).

.grid_remove()

Con este otro método eliminaremos el widget de la pantalla. Al contrario que con .grid_forget() si recordará las opciones de posicionamiento. Con .grid() el objeto volverá a su posición original.

.grid_size()

Llamando a esta rutina sin argumentos nos devolverá el tamaño del widget. Este tamaño vendrá dado en número de filas y de columnas ocupadas.

.grid_slaves(row, column)

Invocando a esta función sin valores nos devolverá los widgets heredados del objeto selecionado. También podemos pedirle que nos devuelva solo los pertenecientes a una determinada fila o columna. Para eso tendremos que pasarle el valor de posición como argumento de la rutina.

Forzando el tamaño de columnas y filas

Aunque con los métodos anteriores podemos fijar la posición y el tamaño de los objetos aún no hemos explicado como modificar el tamaño de las columnas y filas.

El método grid reserva una rutina para configurar algunos parámetros. Su estructura es la siguiente:

.columnconfigure(N, opcion = valor)

Siendo N el valor de la columna a modificar, las opciones se fijarán a continuación.

.rowconfigure(N, opcion = valor)

Siendo N el valor de la fila a configurar, los parámetros se configuran a continuación.

Las opciones que podremos configurar serán las tres siguientes:

- "minsize". Tamaño mínimo de la columna o fila a definir. Esta opción solo tendrá efecto cuando haya algún objeto en esa columna o fila.
- "pad". Margen, en píxeles, a añadir a la columna o fila más alla del objeto más grande que contengan.
- "weight". Esta opción nos permitirá hacer la ventana estirable. Para conseguir esto tendremos que comunicarle al sistema el incremento relativo de píxeles de las columnas o filas al aumentar el tamaño de la ventana. Poniendo un ejemplo con columnas:
  - columnconfigure(0, weight = 3)
  - columnconfigure(1, weight = 1)

  La primer columna, la número cero, aumentará tres veces más de tamaño que la segunda columna, la número uno.

Ejemplo práctico

Para ilustrar el funcionamiento de este método pondremos un ejemplo práctico.
En ese código fuente ajustaremos los más importantes parámetros de este método.

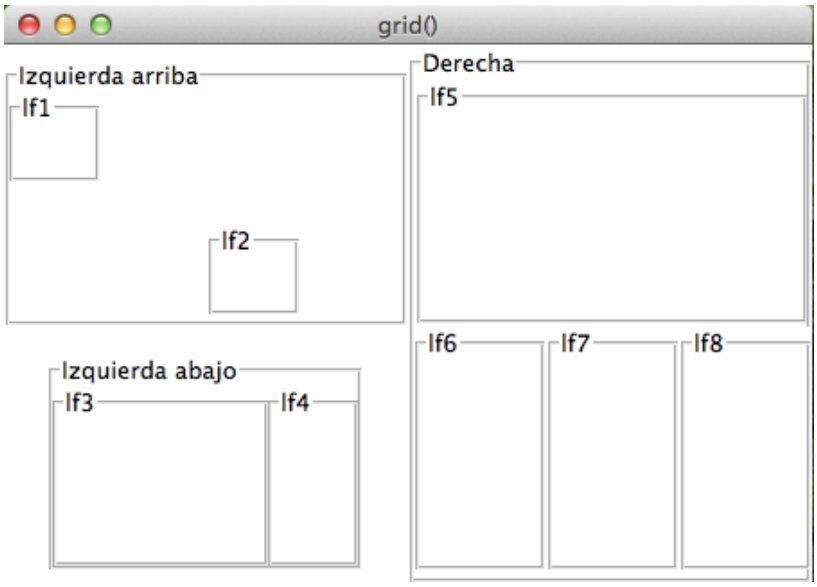


Figura 47. Ejemplo de empaquetado con el método grid.

Debido a la extensión del código dividiremos la explicación en cuatro partes. Primero explicaremos la creación de la ventana y después el contenido de los tres LabelFrames que la componen. Expondremos el código fuente parte a parte.

```
1  import Tkinter

2  root = Tkinter.Tk()
3  root.title("grid()")
4  root.geometry("450x300")
```

Una vez importado el módulo Tkinter (línea primera) se pasa a la creación de la ventana. Esta se realiza como es habitual creando un objeto de la clase Tk() del módulo Tkinter.
En la línea número tres damos a la ventana el nombre de grid(), dándole un tamaño de 450 píxeles de ancho y 300 píxeles de alto en la línea cuatro.

```
5  lf_izquierdaarriba = Tkinter.LabelFrame(root, text = "Izquierda
arriba", width = 223, height = 148)
6  lf_izquierdaarriba.grid(column = 0, row = 0, columnspan = 1,
rowspan = 1, padx = 1, pady = 1)
7  lf_izquierdaarriba.grid_propagate(0)
8  lf_izquierdaarriba.columnconfigure(0, minsize = 111)
9  lf_izquierdaarriba.columnconfigure(1, minsize = 111)

10 lf_izquierdaarriba1 = Tkinter.LabelFrame(lf_izquierdaarriba, text
= "lf1", width = 50, height = 50)
11 lf_izquierdaarriba1.grid(column = 0, row = 0, columnspan = 1,
rowspan = 1, sticky = Tkinter.NW)

12 lf_izquierdaarriba2 = Tkinter.LabelFrame(lf_izquierdaarriba, text
= "lf2", width = 50, height = 50)
13 lf_izquierdaarriba2.grid(column = 1, row = 1, columnspan = 1,
rowspan = 1, sticky = Tkinter.NW)
```

El labelframe con la etiqueta “Izquierda arriba” y sus elementos heredados se encuentran definidos en las líneas que van de la número cinco a la numero trece.
Este LabelFrame está ubicado en la primera columna y la primera fila, así que solo ocupa una celda.

En este ejemplo hemos tratado de explicar la utilidad de forzar el tamaño de las columnas.

Como podemos comprobar, dentro del LabelFrame “lf_izquierdaarriba” tenemos dos LabelFrame (lf1 y lf2) de un tamaño bastante inferior al de su contenedor.
Por defecto, “lf_izquierdaarriba” tendría que ajustarse al tamaño de “lf1” y “lf2”. Esto no es así porque hemos configurado el tamaño mínimo de las dos columnas a la mitad del tamaño total del objeto.

Nos sobra espacio en las dos columnas, así que con el parámetro "sticky" hemos fijado la posición de los dos elementos heredados. Ambos se encuentran en la posición noroeste de su respectiva celda.

El resto de opciones de configuración (column, row, columnspan, rowspan, …) son las usuales. Ya están explicadas en el punto anterior, así que no entraremos en su definición.

```
14 lf_izquierdaabajo = Tkinter.LabelFrame(root, text = "Izquierda
abajo", width = 223, height = 148)
15 lf_izquierdaabajo.grid(column = 0, row = 1, columnspan = 1,
rowspan = 1, padx = 1, pady = 1)

16 lf_izquierdaabajo1 = Tkinter.LabelFrame(lf_izquierdaabajo, text =
"lf3", width = 120, height = 100)
17 lf_izquierdaabajo1.grid(column = 0, row = 0, columnspan = 1,
rowspan = 1)

18 lf_izquierdaabajo2 = Tkinter.LabelFrame(lf_izquierdaabajo, text =
"lf4", width = 50, height = 100)
19 lf_izquierdaabajo2.grid(column = 1, row = 0, columnspan = 1,
rowspan = 1)
```

Entre la línea número catorce y la línea número diecinueve se encuentra definido el segundo LabelFrame de esta ventana. Este se encuentra en la segunda fila y la primera columna. Al igual que el anterior solo ocupa una celda.
En este widget hemos tratado de exponer el caso contrario al objeto anterior.

No hemos forzado ningún tamaño de columnas ni de filas y no hemos hecho uso del método .grid_propagate().
Esto nos lleva a que el tamaño del LabelFrame "lf_izquierdaabajo" se ha ajustado al tamaño de los elementos que contiene.
También comprobamos que el tamaño de las dos columnas internas del LabelFrame, se ajusta al distinto tamaño de sus dos elementos (lf3 y lf4).

Los parámetros de configuración de los LabelFrame son los mismos que el objeto anterior, así que su comprensión no debería representar ninguna dificultad.

```
20 lf_derecha = Tkinter.LabelFrame(root, text = "Derecha", width =
223, height = 298)
21 lf_derecha.grid(column = 1, row = 0, columnspan = 1, rowspan = 2,
padx = 1, pady = 1)
22 lf_derecha.grid_propagate(0)

23 lf_derecha1 = Tkinter.LabelFrame(lf_derecha, text = "lf5", width =
217, height = 135)
24 lf_derecha1.grid(column = 0, row = 0, columnspan = 3, rowspan = 1,
padx = 1, pady = 1)
25 lf_derecha1.grid_propagate(0)
```

```
26 lf_derecha2 = Tkinter.LabelFrame(lf_derecha, text = "lf6", width =
72, height = 135)
27 lf_derecha2.grid(column = 0, row = 1, columnspan = 1, rowspan = 1,
padx = 1, pady = 1)
28 lf_derecha2.grid_propagate(0)

29 lf_derecha3 = Tkinter.LabelFrame(lf_derecha, text = "lf7", width =
72, height = 135)
30 lf_derecha3.grid(column = 1, row = 1, columnspan = 1, rowspan = 1,
padx = 1, pady = 1)
31 lf_derecha3.grid_propagate(0)

32 lf_derecha4 = Tkinter.LabelFrame(lf_derecha, text = "lf8", width =
72, height = 135)
33 lf_derecha4.grid(column = 2, row = 1, columnspan = 1, rowspan = 1,
padx = 1, pady = 1)
34 lf_derecha4.grid_propagate(0)
```

El tercer, y último, LabelFrame de la ventana y sus widgets se encuentran entre la línea veinte y la línea treinta y cuatro.
La LabelFrame

```
35 root.mainloop()
```

Como es habitual en los códigos de ejemplo de esta memoria, el código fuente del script termina con la orden de arranque de la ventana.

Croquis de la ventana principal

Una vez definidos los métodos para la creación de los elementos de la ventana principal pasaremos a definir el croquis de los elementos de esta.
Como hemos mencionado anteriormente agruparemos los elementos por medio de labelframes. Los grupos creados tendrán los elementos detallados a continuación:

- Menú para seleccionar, mostrar y/o modificar un observador.
    - La modificación de la ubicación requerirá de demasiados elementos para englobarla dentro de esté widget así que irá en una ventana aparte.
    - Para seleccionar la ubicación usaremos un widget de la clase ScrollBar.
    - Mediante etiquetas se mostrará de que tipo es cada valor del menú.
    - Usando objetos del tipo stringvar dentro de etiquetas se mostrarán los valores de la localización seleccionada.
    - Los datos que hemos de mostrar acerca de cada localización son:
        - Nombre.
        - Latitud.
        - Longitud.
        - Altura.
        - Horizonte.
- Menú para mostrar las coordenadas celestes del satélite elegido.

    - Los widgets para este menú serán los habituales. Usaremos labels para mostrar la información, utilizando stringvars para los valores variables.
    - Los datos que tendremos que poner en pantalla serán los siguientes:
        - Ascensión recta.
        - Declinación.
        - Altura.
        - Acimut.
        - Próximo orto.
- Menú para seleccionar un satélite disponible.
    - Tanto la selección de la familia del satélite como la selección del satélite mismo se hará por medio de un widget de la clase Scrollbar.
    - El texto de la Scrollbar se modificará de forma dinámica.
- Menú para mostrar y actualizar el tiempo del sistema
    - Mediante stringvars se mostrará la hora y fecha del sistema.
    - Un objeto de la clase Button servirá para llamar a la rutina de actualización del tiempo del sistema.
- Menú para configurar diferentes opciones de funcionamiento del programa y para activar las rutinas de seguimiento.
    - Las diferentes opciones de funcionamiento del programa se configurarán mediante una notebook. La gran ventaja de esta es que en un pequeño espacio se pueden colocar diferentes paneles.
    - La manera más sencilla para interactuar con las rutinas de seguimiento del programa es el uso de botones. Tendremos que crear botones para las siguientes funciones del programa:
        - Mostrar la posición actual de un objeto.
        - Activar el seguimiento.
        - Detener el seguimiento.
        - Acceder al menú de configuración.
        - Salir del programa.

La posición de los elementos mencionados anteriormente se detallará en el punto siguiente.

Esquema inicial

Atendiendo a las especificaciones del ejemplo anterior podemos crear un pequeño esquema de nuestra interfaz principal.
En este esquema detallaremos someramente donde irá cada elemento.

Capturas en Mac OS X

La apariencia de nuestro código en Mac OS X será la siguiente:

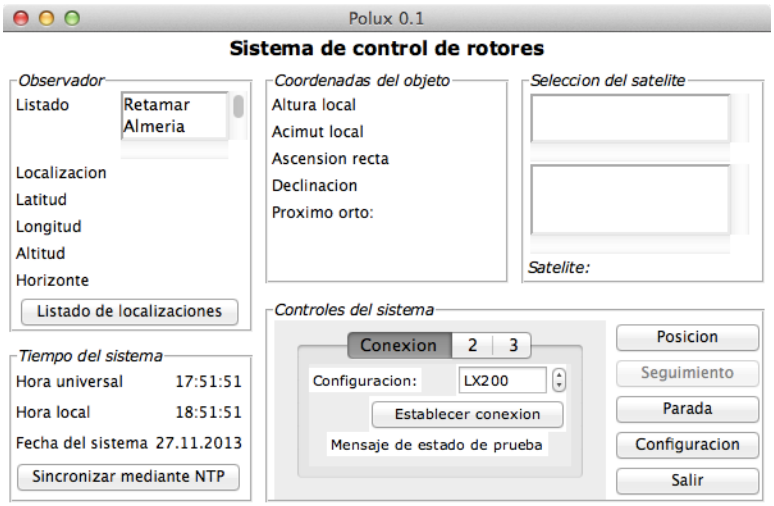


Figura 48. Ventana principal en Mac OS X.

Como podemos ver, todos los elementos se encuentran en la posición deseada. El diseño por defecto del sistema da una interfaz bastante limpia, así que no realizaremos ningún cambio.

Modificando el código para Linux

El mismo código en una distro de linux, en este caso Raspbian, nos dará la siguiente imagen:

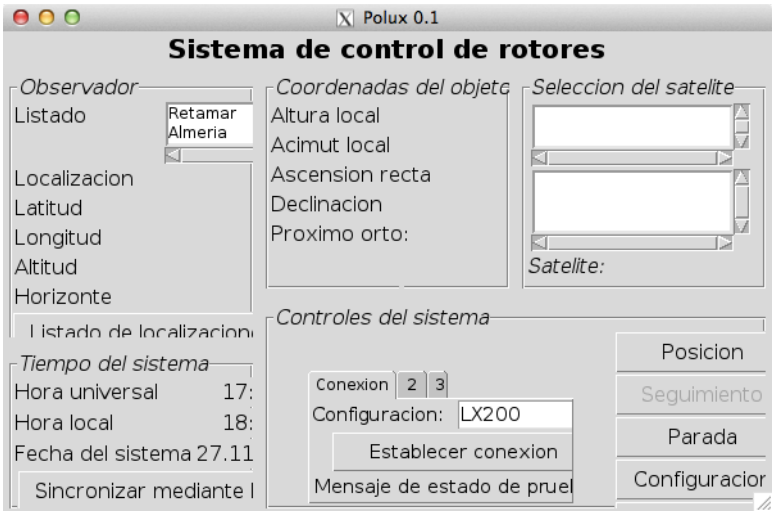


Figura 49. Ventana principal del script en Raspbian.

El aspecto del programa en este caso cambia. Esto es debido a el sistema de ventanas X11 no posee algunas funcionalidades como el suavizado de fuentes.
El aspecto de la Notebook también cambia ya que, al no haber definido ningún diseño, toma el diseño por defecto de X11 y este es muy diferente al diseño predefinido de los sistemas Macintosh.

Como primera aproximación al problema podemos variar el tamaño de la letra.

Al haber definido el diseño de las letras mediante tipografías solo tendremos que variar las tipografías del sistema. Esto se podrá hacer con un simple if que, dependiendo del sistema operativo, asigne un tamaño a las letras de las tipografías del sistema.
Nuestro código al crear las tipografias será el siguiente:

```
# Tipografia para Mac OS
1 if sys.platform == 'darwin':
2       self.Verd15Bold = tkFont.Font(family = 'Verdana', size = '15',
weight = 'bold')
3       self.Verd12Italic = tkFont.Font(family = 'Verdana', size =
'12', slant = 'italic')
4       self.Verd13Italic = tkFont.Font(family = 'Verdana', size =
'13', slant = 'italic')
5       self.Verd12Roman = tkFont.Font(family = 'Lucida', size = '12',
slant = 'roman')
6       self.Verd11Roman = tkFont.Font(family = 'Verdana', size =
'11', slant = 'roman')

# Tipografia para Linux
7  elif sys.platform.startswith('linux'):
8      self.Verd15Bold = tkFont.Font(family = 'Verdana', size = '13',
weight = 'bold')
9      self.Verd12Italic = tkFont.Font(family = 'Verdana', size =
'10', slant = 'italic')
10     self.Verd13Italic = tkFont.Font(family = 'Verdana', size =
'11', slant = 'italic')
11     self.Verd12Roman = tkFont.Font(family = 'Lucida', size = '10',
slant = 'roman')
12     self.Verd11Roman = tkFont.Font(family = 'Verdana', size = '9',
slant = 'roman')
```

Como podemos comprobar, el tamaño de las tipografías creadas en las líneas dos, tres, cuatro, cinco y seis se ha reducido dos puntos en las líneas ocho, nueve, diez, once y doce.

Estas modificaciones nos darán el siguiente resultado.

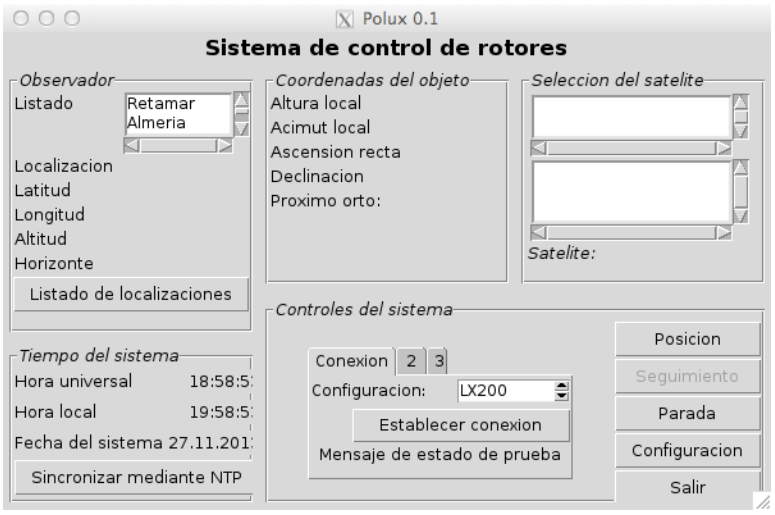


Figura 50. Primera modificación del código de la ventana principal en Linux.

Aún persisten algunos problemas como son: el tamaño del objeto Notebook y los objetos de la LabelFrame “Tiempo del sistema”.

Primero entraremos con las dificultades encontradas en la LabelFrame.
Como podemos comprobar, la etiqueta “Fecha del sistema” y la fecha actual no entran en la ventana, aún habiendo reducido el tamaño de la tipografía.
La solución a esto vendrá reduciendo el texto de la etiqueta a “Fecha” y ajustando el tamaño del botón de ese Labelframe. El código fuente para ajustar el texto será el siguiente:

```
# Fecha local
 if sys.platform == 'darwin':
     etiqueta_fecha = Tkinter.Label(ventana_tiempo, text = 'Fecha
del sistema', font = self.Verd12Roman)
     etiqueta_fecha.grid(row = 2, column = 0, rowspan = 1,
columnspan = 1, sticky = Tkinter.W)
     etiqueta_fecha.grid_propagate(0)
 elif sys.platform.startswith('linux'):
     etiqueta_fecha = Tkinter.Label(ventana_tiempo, text = 'Fecha',
font = self.Verd12Roman)
     etiqueta_fecha.grid(row = 2, column = 0, rowspan = 1,
columnspan = 1, sticky = Tkinter.W)
     etiqueta_fecha.grid_propagate(0)
 else:
     print "Sistema no soportado"
```

Para modificar el tamaño del botón reduciremos la cantidad de caracteres que lleva escrito. “Sincronizar mediante NTP” se convertirá en “Sincronizar por NTP”. Esta modificación se llevará a cabo con el siguiente código:

```
# Boton para sincronizar la hora
  if sys.platform == "darwin":
     boton_sincronizar = Tkinter.Button(ventana_tiempo, text =
'Sincronizar mediante NTP', font = self.Verd12Roman, command =
self.sincronizar_hora)
     boton_sincronizar.grid(row = 3, column = 0, rowspan = 1,
columnspan = 2, sticky = Tkinter.S)
     boton_sincronizar.grid_propagate(0)
  elif sys.platform.startswith('linux'):
     boton_sincronizar = Tkinter.Button(ventana_tiempo, text =
'Sincronizar por NTP', font = self.Verd12Roman, command =
self.sincronizar_hora, width = 19)
     boton_sincronizar.grid(row = 3, column = 0, rowspan = 1,
columnspan = 2, sticky = Tkinter.S)
     boton_sincronizar.grid_propagate(0)
  else:
     print "Sistema no soportado"
```

Para ajustar el Notebook tendremos que variar el tamaño de este. Ajustaremos un tamaño de 105 píxeles de alto por 250 píxeles de ancho, para ello utilizaremos el código siguiente:

```
1  # Creamos la notebook
2  if sys.platform == "darwin":
3      estados_notebook = ttk.Notebook(ventana_controles, height =
80, width = 205)
4      estados_notebook.grid(row = 0, column = 0, rowspan = 5,
columnspan = 1)
5      estados_notebook.grid_propagate(0)
6 elif sys.platform.startswith('linux'):
7      estados_notebook = ttk.Notebook(ventana_controles, height =
105, width = 250)
8      estados_notebook.grid(row = 0, column = 0, rowspan = 5,
columnspan = 1)
9      estados_notebook.grid_propagate(0)
10  else:
11     print "Sistema no soportado"
```

Todos estos cambios nos darán la siguiente salida por pantalla:

Figura 51. Segunda modificación del código de la ventana principal en Linux.

Como podemos comprobar, el único elemento disonante de la interfaz son los cinco botones de la derecha.
Ya que no podemos variar el alto de los botones pasaremos el botón "Configuración" al objeto Notebook.
Lo introduciremos a continuación del mensaje de estado. De paso modificaremos la disposición de los elementos del Notebook para así mejorar su aspecto.

Lo primero que haremos será modificar el tamaño mínimo de las columnas para desplazar los elementos de la columna segunda a la derecha. Esto lo conseguiremos fijando un ancho mínimo de la primera columna a un valor más alto.
El añadido al código será el siguiente:

```
1  primera_ventana.columnconfigure(0, minsize = 130)
2  primera_ventana.columnconfigure(1, minsize = 80)
```

El siguiente paso será colocar el botón dentro del widget. Para ello crearemos un nuevo if. Este colocará el boton en una u otra posición dependiendo del sistema operativo utilizado.

```
# Boton de configuracion
1  if sys.platform == 'darwin':
2     self.boton_configuracion = Tkinter.Button(ventana_controles,
text = 'Configuracion', font = self.Verd12Roman, height = 1, width =
ancho, command = self.configurar_sistema)
3     self.boton_configuracion.grid(row = 3, column = 1)
4  elif sys.platform.startswith('linux'):
5     self.boton_configuracion = Tkinter.Button(primera_ventana, text
= 'Configuracion', font = self.Verd12Roman, height = 1, width =
ancho, command = self.configurar_sistema)
6     self.boton_configuracion.grid(row = 3, column = 1, rowspan = 1,
columnspan = 1)
```

La salida por pantalla de las modificaciones anteriores será la mostrada a continuación:

Figura 52. Tercera, y última, modificación del código de la ventana principal en Linux.

El aspecto general de la ventana, aunque distinto en algunos puntos al ofrecido en Mac OS X, será totalmente válido para nuestro cometido.

Una vez corregido el código fuente para su uso en Linux pasaremos a los sistemas operativos de la familia Windows.

Modificando el código para Windows

La primera captura de pantalla que obtendremos, en Windows XP, con el código del punto anterior será esta:

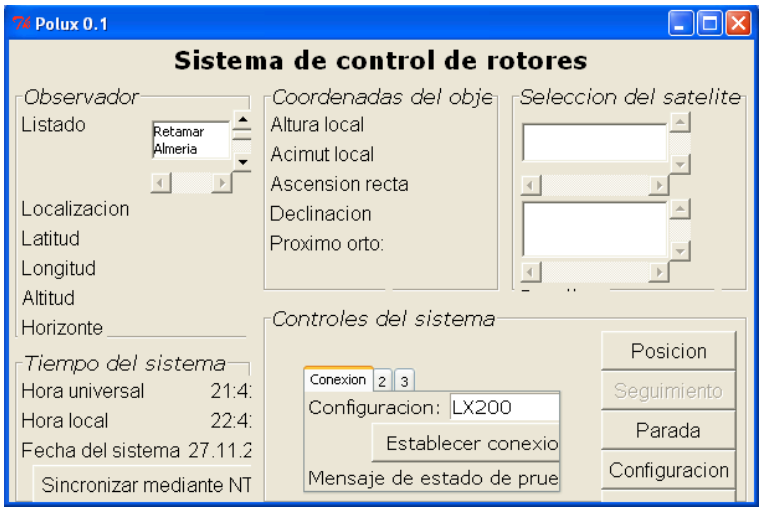


Figura 53. Ventana principal del script en Windows XP.

Como podemos ver tenemos algunos problemas similares al ejemplo anterior.
Empezaremos creando otra tipografía para nuestro sistema en Windows. Esto se hará añadiendo otro "elif" a nuestro if-elif-else inicial.

```
# Tipografia para Mac OS X
1   if sys.platform == 'darwin':
2       self.Verd15Bold = tkFont.Font(family = 'Verdana', size = '15',
weight = 'bold')
3       self.Verd12Italic = tkFont.Font(family = 'Verdana', size =
'12', slant = 'italic')
4       self.Verd13Italic = tkFont.Font(family = 'Verdana', size =
'13', slant = 'italic')
5       self.Verd12Roman = tkFont.Font(family = 'Lucida', size = '12',
slant = 'roman')
6       self.Verd11Roman = tkFont.Font(family = 'Verdana', size = '11',
slant = 'roman')
# Tipografia para Linux
7   elif sys.platform.startswith('linux'):
8       self.Verd15Bold = tkFont.Font(family = 'Verdana', size = '13',
weight = 'bold')
9       self.Verd12Italic = tkFont.Font(family = 'Verdana', size =
'10', slant = 'italic')
10      self.Verd13Italic = tkFont.Font(family = 'Verdana', size =
'11', slant = 'italic')
11      self.Verd12Roman = tkFont.Font(family = 'Lucida', size = '10',
slant = 'roman')
12      self.Verd11Roman = tkFont.Font(family = 'Verdana', size = '9',
slant = 'roman')
# Tipografia para Windows
13  elif sys.platform == 'win32':
14      self.Verd15Bold = tkFont.Font(family = 'Verdana', size = '13',
weight = 'bold')
15      self.Verd12Italic = tkFont.Font(family = 'Verdana', size =
'10', slant = 'italic')
16      self.Verd13Italic = tkFont.Font(family = 'Verdana', size =
'11', slant = 'italic')
```

```
17     self.Verd12Roman = tkFont.Font(family = 'Lucida', size = '10',
slant = 'roman')
18     self.Verd11Roman = tkFont.Font(family = 'Verdana', size = '9',
slant = 'roman')
19 else:
20     print "Sistema no soportado"
```

Las líneas uno a doce serán las mismas que en el punto anterior. De la línea número trece a la línea dieciocho introduciremos nuestros cambios.

Nos damos cuenta también de que el tamaño del widget Notebook no es el correcto. Fijaremos el mismo tamaño que en Linux y, a continuación, lo ajustaremos si no es el correcto. El código alterado serán el siguiente:

```
# Creamos la Notebook
1   if sys.platform == "darwin":
2      estados_notebook = ttk.Notebook(ventana_controles, height = 80,
width = 205)
3      estados_notebook.grid(row = 0, column = 0, rowspan = 5,
columnspan = 1)
4      estados_notebook.grid_propagate(0)
5   elif sys.platform.startswith('linux'):
6      estados_notebook = ttk.Notebook(ventana_controles, height =
105, width = 250)
7      estados_notebook.grid(row = 0, column = 0, rowspan = 5,
columnspan = 1)
8      estados_notebook.grid_propagate(0)
9   elif sys.platform == 'Win32':
10     estados_notebook = ttk.Notebook(ventana_controles, height =
105, width = 250)
11     estados_notebook.grid(row = 0, column = 0, rowspan = 5,
columnspan = 1)
12     estados_notebook.grid_propagate(0)
13 else:
14     print "Sistema no soportado"
```

El código es el mismo que el punto anterior así que no necesitaremos entrar en más detalles.

También tendremos que modificar la etiqueta “Fecha del sistema” para substituirla, como en el ejemplo anterior, por una más corta, “Fecha”.

```
# Fecha local
1   if sys.platform == 'darwin':
2      etiqueta_fecha = Tkinter.Label(ventana_tiempo, text = 'Fecha
del sistema', font = self.Verd12Roman)
3      etiqueta_fecha.grid(row = 2, column = 0, rowspan = 1,
columnspan = 1, sticky = Tkinter.W)
4      etiqueta_fecha.grid_propagate(0)
5   elif sys.platform.startswith('linux'):
6      etiqueta_fecha = Tkinter.Label(ventana_tiempo, text = 'Fecha',
font = self.Verd12Roman)
```

```
7     etiqueta_fecha.grid(row = 2, column = 0, rowspan = 1,
columnspan = 1, sticky = Tkinter.W)
8     etiqueta_fecha.grid_propagate(0)
9  elif sys.platform == 'win32':
10    etiqueta_fecha = Tkinter.Label(ventana_tiempo, text = 'Fecha',
font = self.Verd12Roman)
11    etiqueta_fecha.grid(row = 2, column = 0, rowspan = 1,
columnspan = 1, sticky = Tkinter.W)
12    etiqueta_fecha.grid_propagate(0)
13 else:
14    print "Sistema no soportado"
```

Los cinco botones de control del LabelFrame “Controles del sistema” no entran dentro del widget base. Haremos igual que en Linux y meteremos el botón configuración dentro del objeto de la clase Notebook.

```
# Boton de configuracion
1  if sys.platform == "darwin":
2     self.boton_configuracion = Tkinter.Button(ventana_controles,
text = 'Configuracion', font = self.Verd12Roman, height = 1, width =
11, command = self.configurar_sistema)
3     self.boton_configuracion.grid(row = 3, column = 1)
4  elif sys.platform.startswith('linux'):
5     self.boton_configuracion = Tkinter.Button(primera_ventana, text
= 'Configuracion', font = self.Verd12Roman, height = 1, width = 11,
command = self.configurar_sistema)
6     self.boton_configuracion.grid(row = 3, column = 1, rowspan = 1,
columnspan = 1)
7  elif sys.platform == "win32":
8     self.boton_configuracion = Tkinter.Button(primera_ventana, text
= 'Configuracion', font = self.Verd12Roman, height = 1, width = 11,
command = self.configurar_sistema)
9     self.boton_configuracion.grid(row = 3, column = 1, rowspan = 1,
columnspan = 1)
10 else:
11    print "Sistema no soportado"
```

Como último paso antes de probar el funcionamiento del nuevo código modificaremos el texto del boton “Sincronizar mediante NTP”. Substituiremos este texto por uno más corto. Reutilizaremos el código del ejemplo anterior.

```
1  # Boton para sincronizar la hora
2  if sys.platform == "darwin":
3     boton_sincronizar = Tkinter.Button(ventana_tiempo, text =
'Sincronizar mediante NTP', font = self.Verd12Roman, command =
self.sincronizar_hora)
4     boton_sincronizar.grid(row = 3, column = 0, rowspan = 1,
columnspan = 2, sticky = Tkinter.S)
5     boton_sincronizar.grid_propagate(0)
6  elif sys.platform.startswith('linux'):
```

```
7     boton_sincronizar = Tkinter.Button(ventana_tiempo, text =
'Sincronizar por NTP', font = self.Verd12Roman, command =
self.sincronizar_hora, width = 19)
8     boton_sincronizar.grid(row = 3, column = 0, rowspan = 1,
columnspan = 2, sticky = Tkinter.S)
9     boton_sincronizar.grid_propagate(0)
10 elif sys.platform == "win32":
11    boton_sincronizar = Tkinter.Button(ventana_tiempo, text =
'Sincronizar por NTP', font = self.Verd12Roman, command =
self.sincronizar_hora, width = 19)
12    boton_sincronizar.grid(row = 3, column = 0, rowspan = 1,
columnspan = 2, sticky = Tkinter.S)
13    boton_sincronizar.grid_propagate(0)
14 else:
15    print "Sistema no soportado"
```

La salida por pantalla de nuestro sistema será la mostrada a continuación.

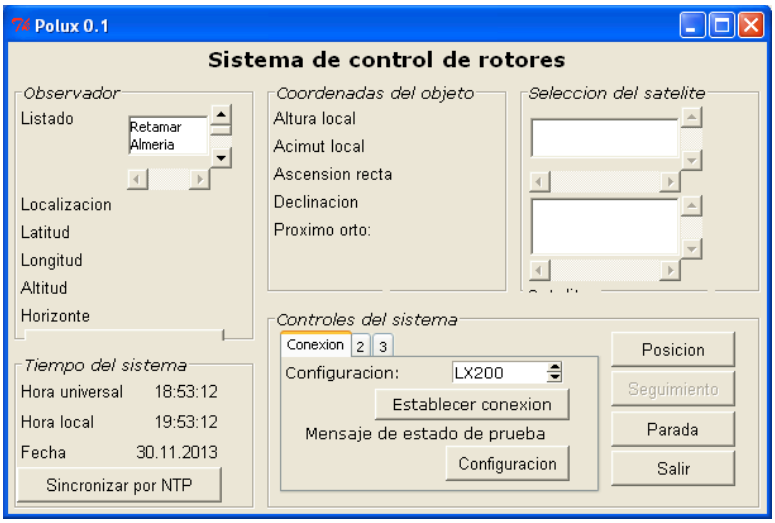


Figura 54. Segunda modificación del código de la ventana principal en Windows XP.

Podemos comprobar que el botón para acceder al listado de localizaciones no resulta visible en Windows XP. Tampoco entra en pantalla la etiqueta para mostrar el satélite seleccionado (en el Labelframe "Selección del satélite").
Para solucionar esto tendremos que realizar dos acciones:

1. Eliminar la etiqueta "Horizonte" y su etiqueta asociada para mostrar el horizonte.
2. Pasar la etiqueta para mostrar el satélite actual al Labelframe "Coordenadas del objeto".

El primer punto solo consistirá en crear un if-elif-else que evite la creación de la etiqueta.

```
1  if sys.platform == "darwin":
2     # Etiqueta 'Horizonte'
3     etiqueta_horizonte = Tkinter.Label(ventana_observador, text =
'Horizonte', font = self.Verd12Roman)
etiqueta_horizonte.grid(row = 5, column = 0, rowspan = 1, columnspan
= 1, sticky = Tkinter.W)
```

```
4      etiqueta_horizonte.grid_propagate(0)
5      # Creacion de la StringVar para la altitud
6      self.text_var_horizontebueno = Tkinter.StringVar()
7      self.text_var_horizontebueno.set('')
8      # Etiqueta para mostrar el horizonte
9      mostrar_horizonte = Tkinter.Label(ventana_observador,
textvariable = self.text_var_horizontebueno, font = self.Verd12Roman)
10     mostrar_horizonte.grid(row = 5, column = 1, rowspan = 1,
columnspan = 1, sticky = Tkinter.E)
11     mostrar_horizonte.grid_propagate(0)
12     row_nuevo = 6
13  elif sys.platform.startswith('linux')
14     # Etiqueta 'Horizonte'
15     etiqueta_horizonte = Tkinter.Label(ventana_observador, text =
'Horizonte', font = self.Verd12Roman)
16     etiqueta_horizonte.grid(row = 5, column = 0, rowspan = 1,
columnspan = 1, sticky = Tkinter.W)
17     etiqueta_horizonte.grid_propagate(0)
18     # Creacion de la StringVar para la altitud
19     self.text_var_horizontebueno = Tkinter.StringVar()
20     self.text_var_horizontebueno.set('')
21     # Etiqueta para mostrar el horizonte
22     mostrar_horizonte = Tkinter.Label(ventana_observador,
textvariable = self.text_var_horizontebueno, font = self.Verd12Roman)
23     mostrar_horizonte.grid(row = 5, column = 1, rowspan = 1,
columnspan = 1, sticky = Tkinter.E)
24     mostrar_horizonte.grid_propagate(0)
25     row_nuevo = 6
26  else:
27     row_nuevo = 5
```

Como vemos, en el código, se ha creado una nueva variable (row_nuevo) cuyo objetivo es fijar la posición del siguiente elemento en el LabelFrame.

Al eliminar una etiqueta que realizaba acciones del script tendremos que eliminar todas las referencias a ella. Así evitaremos que el programa caiga en una excepción.

Modificaremos ahora el código de la dos etiquetas mencionadas anteriormente para mostrarlas en la nueva ubicación. Tendremos que cambiar gran parte del código del Labelframe en cuestión. Debido a la gran extensión del código, y a que el código para Mac OS X y linux no se verá modificado, solo mencionaremos la parte que ataña a Windows.

```
1   elif sys.platform == "win32":
2   # Etiqueta 'Proximo orto'
3      etiqueta_orto = Tkinter.Label(ventana_localizacion, text =
'Proximo orto:', font = self.Verd12Roman)
4      etiqueta_orto.grid(row = 4, column = 0, rowspan = 1, columnspan
= 2, sticky = Tkinter.W)
5      etiqueta_orto.grid_propagate(0)
6      # Creacion de la StringVar para el orto
7      self.text_var_ortobueno = Tkinter.StringVar()
8      self.text_var_ortobueno.set('')
```

```
9       # Etiqueta para mostrar el orto
10      mostrar_orto = Tkinter.Label(ventana_localizacion, textvariable
= self.text_var_ortobueno, font = self.Verd12Roman)
11      mostrar_orto.grid(row = 4, column = 0, rowspan = 1, columnspan
= 2, sticky = Tkinter.E)
12      mostrar_orto.grid_propagate(0)
13      # Creacion de la StringVar para eventos
14      self.text_var_eventobueno = Tkinter.StringVar()
15      self.text_var_eventobueno.set('')
16      # Etiqueta para mostrar el evento
17      mostrar_evento = Tkinter.Label(ventana_localizacion,
textvariable = self.text_var_eventobueno, font = self.Verd12Roman)
mostrar_evento.grid(row = 5, column = 0, rowspan = 1, columnspan = 2)
19      mostrar_evento.grid_propagate(0)
18      # Etiqueta 'Satelite'
19      etiqueta_satelite = Tkinter.Label(ventana_localizacion, text =
'Satelite:', font = self.Verd12Roman)
20      etiqueta_satelite.grid(row = 6, column = 0, rowspan = 1,
columnspan = 1, sticky = Tkinter.W)
21      etiqueta_localizacion.grid_propagate(0)
22      # Creacion de la StringVar para mostrar el satelite
23      text_var_satelite = ''
24      self.text_var_satelitebueno = Tkinter.StringVar()
25      self.text_var_satelitebueno.set(text_var_satelite)
26      # Etiqueta para mostrar el satelite
27      mostrar_satelite = Tkinter.Label(ventana_localizacion,
textvariable = self.text_var_satelitebueno, font = self.Verd12Italic)
28      mostrar_satelite.grid(row = 6, column = 1, rowspan = 1,
columnspan = 1, sticky = Tkinter.E)
29      mostrar_satelite.grid_propagate(0)
```

Con este código reubicamos los elementos del Labelframe para que ocupen una fila menos. Así podemos colocar el nuevo elemento dentro del objeto.
La salida por pantalla del código modificado será la mostrada a continuación.

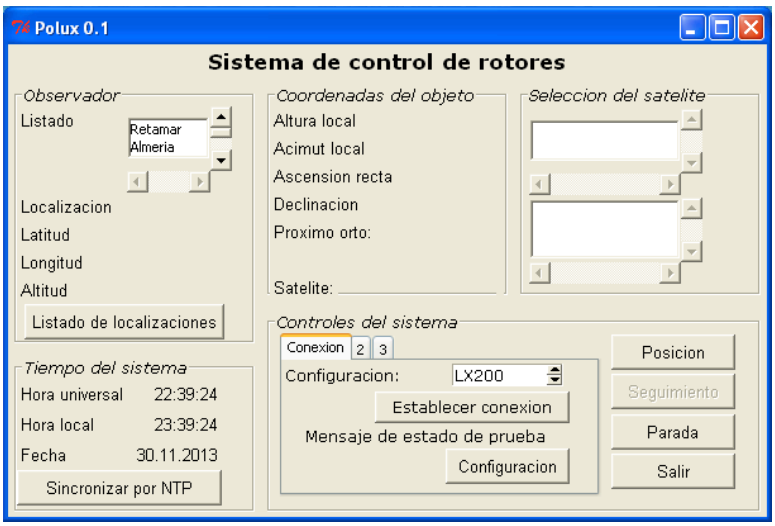


Figura 55. Tercera, y última, modificación del código de la ventana principal en Windows XP.

Con esto damos por concluída la creación de la interfaz de pantalla para la ventana principal.

A continuación probaremos el código fuente de las ventanas restantes del script. Realizaremos las modificaciones necesarias para garantizar la legitibilidad de los elementos del programa.

La ventana de configuración

Croquis de la ventana de configuración

En la ventana de configuración se fijarán los parámetros de funcionamiento del script.
Esta ventana estará dividida en varios paneles mediante un widget del tipo Notebook.
El uso de esta clase de objeto nos permitirá introducir más información en una misma ventana.

En nuestro caso particular usaremos, en un principio, un Notebook de dos paneles con la siguiente información:

1. En el primer panel realizaremos la selección de los TLEs. El script nos dejará ver que satélites componen cada fichero. También podremos forzar la descarga de nuevos elementos.
2. En el segundo panel definiremos los parámetros de los diferentes perfiles de conexión del sistema. También tendremos la opción de configurar los diferentes directorios del sistema.

Capturas en Mac OS X

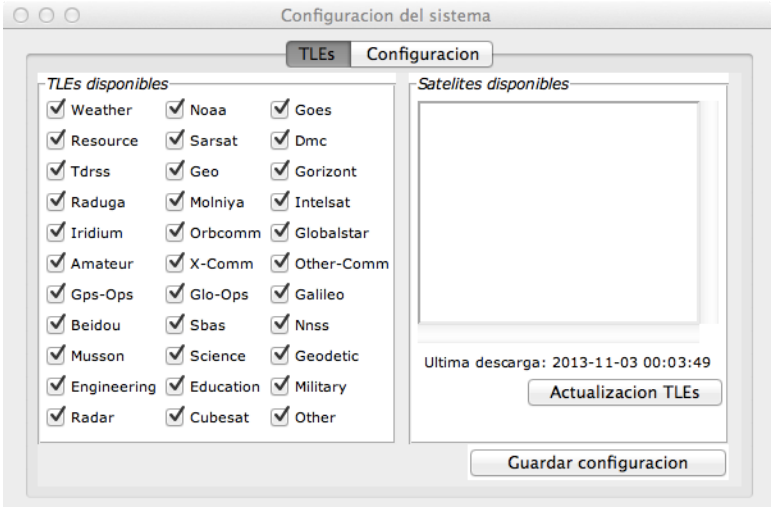


Figura 56. Primer panel de la ventana de configuración en Mac OS X.

El segundo panel lo haremos dependiendo de las necesidades futuras por eso dejaremos espacio disponible.

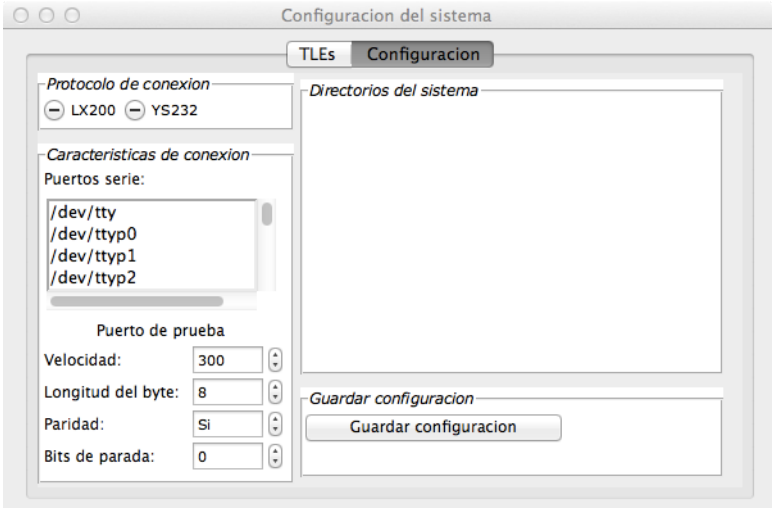


Figura 57. Segundo panel de la ventana de configuración en Mac OS X.

Como podemos comprobar todos los elementos están colocados en la posición especificada por el esquema inicial. Pasaremos ahora a comprobar el funcionamiento del programa en otros sistemas operativos.

Modificando el código para Linux

La ventana de configuración de nuestro script en Raspbian nos dará la siguiente salida por pantalla.

Figura 58. Primer panel de la ventana configuración en Linux.

La Notebook no ocupará toda la pantalla. Esto es debido a que el diseño por defecto de la Notebook en Linux es distinto al diseño preestablecido en Mac OS X.

Comprobaremos si pasa igual en el segundo panel del objeto.

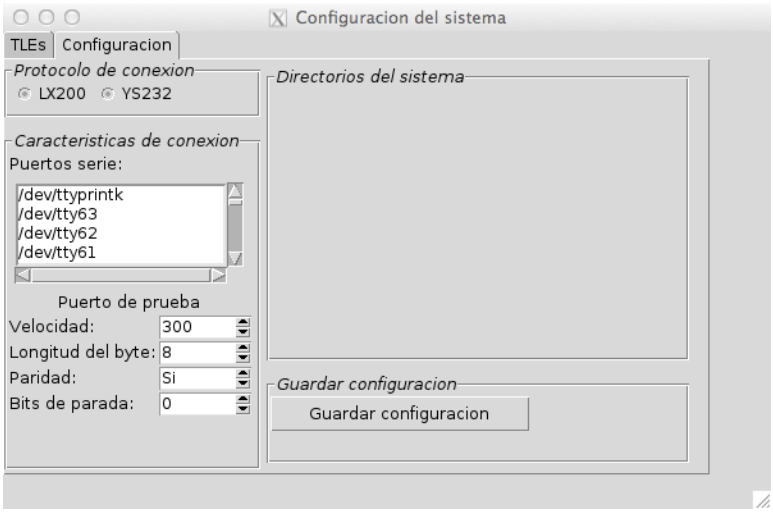


Figura 59. Segundo panel de la ventana configuración en Linux.

En el segundo panel el problema será el mismo que en el primero. La solución más fácil a este será dejar el tamaño de la ventana sin definir.

El ajuste del tamaño se realizará en el momento de la creación de la ventana. El código necesario para ello será el siguiente.

```
1 if sys.platform == "darwin":
2     ventana_configuracion.geometry("600x373+5+5")
3 else:
4     print "Sistema distinto a Mac OS X"
```

Como podemos observar, hemos creado un if-else en el que hemos dejado sin definir el tamaño de la ventana. Así, suponemos, la ventana se ajustará al tamaño del widget que contiene.

Con esta modificación tendremos la siguiente ventana de configuración.

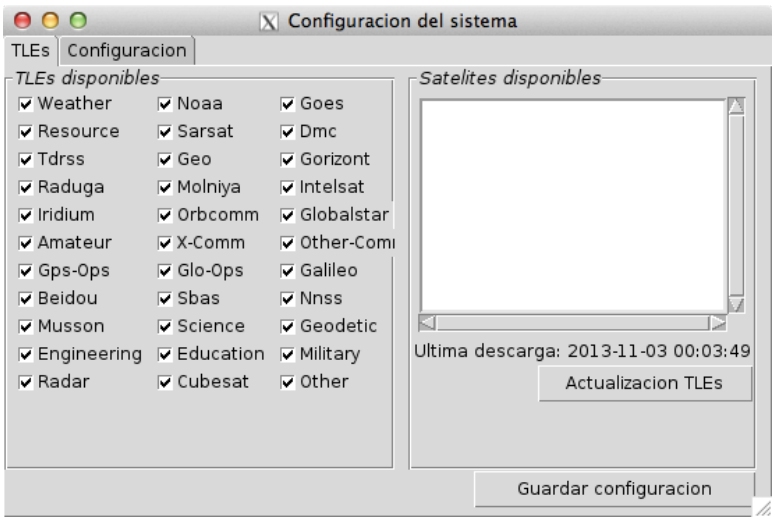


Figura 60. Primera, y última, modificación de la ventana configuración en Linux.

Por las razones comentadas anteriormente el segundo panel tendría que ajustarse también a su ventana. En la siguiente captura de pantalla podemos comprobar este hecho.

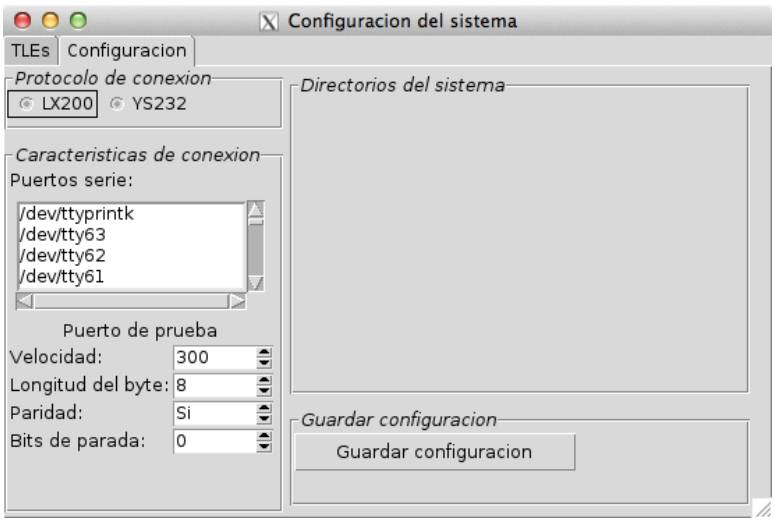


Figura 61. Primera, y última, modificación de la ventana configuración en Linux.

Ajustados los elementos de la ventana configuración para su correcta visualización pasaremos a comprobar el funcionamiento de la ventana Configuración, ahora, en Windows XP.

Modificando del código para Windows

Usaremos el mismo código que en el ejemplo anterior. Supondremos que la ventana también se ajustará al tamaño del widget evitando así que haya espacios vacios en la interfaz.
La captura por pantalla de los paneles de la ventana Configuración serán los siguientes.

Figura 62. Primera, y última, modificacion de la ventana configuración en Windows XP.

El segundo panel por pantalla se verá con el siguiente aspecto.

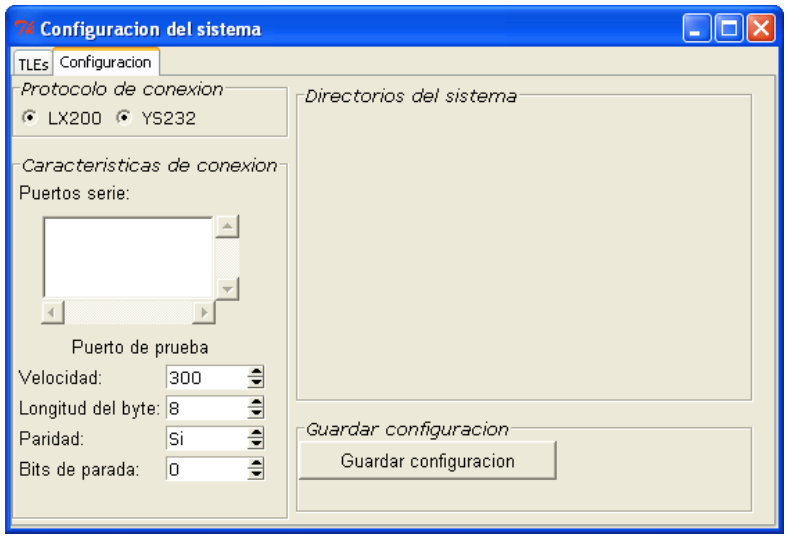


Figura 63. Primera, y última, modificación de la ventana Configuración en Windows XP.

Nuestros planteamientos sobre el tamaño de la ventana, como podemos ver, han sido acertados.
Dando por hecho de que el aspecto de la ventana configuración es el correcto pasaremos a la siguiente ventana del script, la ventana para mostrar y modificar las localizaciones.

El listado de localizaciones

El objetivo de esta ventana será manejar una pequeña base de datos con localizaciones de observadores.
Desde esta ventana tenemos que ser capaces de modificar y eliminar las ubicaciones existentes, además de poder agregar nuevas a la lista.

Capturas en Mac OS X

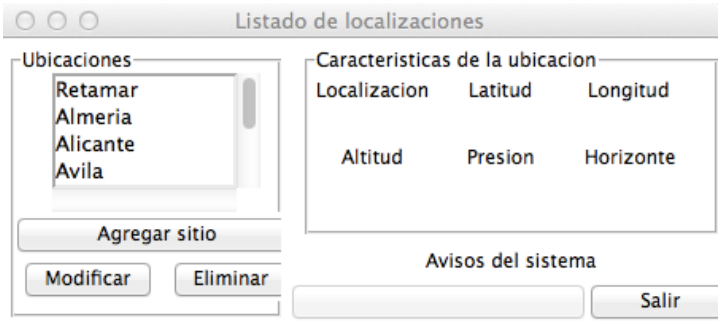


Figura 64. Ventana del listado de localizaciones en Mac OS X.

Como podemos ver el tamaño del botón “Agregar sitio” no es el correcto, siendo más grande lo necesario. Otro botón de la interfaz, el que se encuentra desactivado, también es algo más grande del tamaño necesario.

Reduciremos dos caracteres el botón “Agregar sitio” y dos caracteres el botón desactivado. Este es el código para el primer botón.

```
1  # Boton 'Agregar localizacion'
2  boton_agregar = Tkinter.Button(localizaciones, text = 'Agregar
sitio', font = self.Verd12Roman, width = 16, command =
self.agregar_ubicacion)
3  boton_agregar.grid(row = 1, column = 0, rowspan = 1, columnspan =
2)
```

Siendo este el código para el segundo.

```
1  self.boton_aceptar = Tkinter.Button(self.ventana_posiciones,
textvariable = self.text_var_aceptarconfirmacion, font =
self.Verd12Roman, width = 18, command = self.funcion_variable)
2  self.boton_aceptar.grid(row = 3, column = 1, rowspan = 1,
columnspan = 1)
3  self.boton_aceptar.config(state = Tkinter.DISABLED)
```

Estos dos cambios nos producirán la siguiente salida.

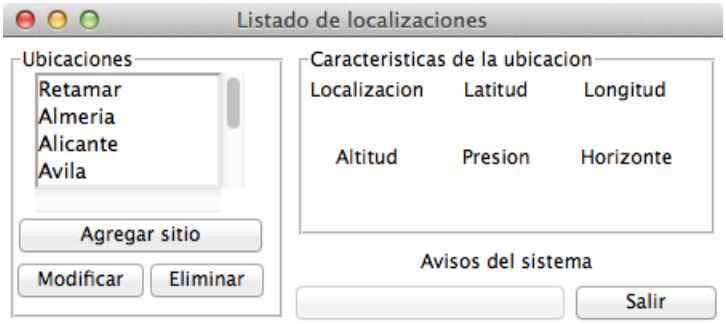


Figura 65. Primera, y última, modificación de la ventana "Listado de localizaciones" en Mac OS X.

La interfaz ya tendrá el aspecto deseado.

Modificando el código para Linux

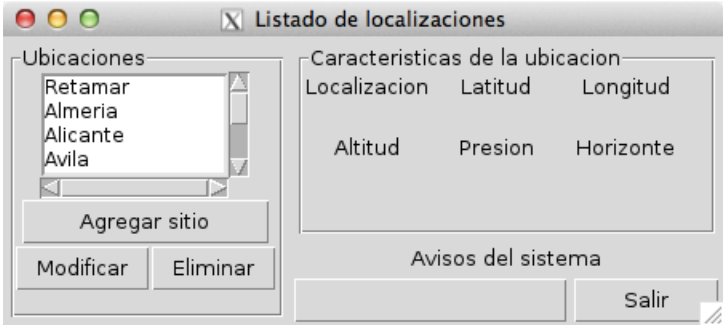


Figura 66. Ventana del listado de localizaciones en Raspbian.

Las modificaciones realizadas a nuestro script también son válidas en Linux como podemos comprobar. Como se puede ver, no hace falta realizar ningún cambio en el código. Pasaremos a continuación a probar el programa en Windows XP.

Modificando del código para Windows

La apariencia de la ventana de este apartado en Windows XP será la siguiente.

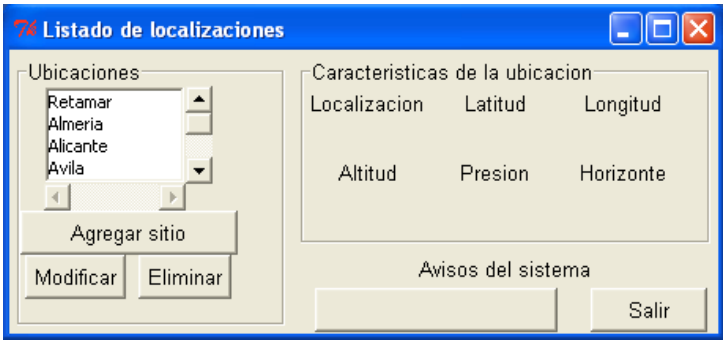


Figura 67. Ventana del listado de localizaciones en Windows XP.

Los elementos de la ventana son perfectamente legibles.

6 – Rutinas de cálculo.

La interfaz gráfica del script no será sino la parte visible de un conjunto de funciones destinadas al correcto funcionamiento del programa.

El hecho de separar las rutinas de control y cálculo de la interfaz gráfica nos permitirá en un futuro, si así lo deseamos, cambiar o, directamente, eliminar la interfaz sin ningún problema.

Una vez explicados ciertos fundamentos astrométricos comenzaremos con un breve repaso de las funciones de cálculo astrométrico utilizadas para después explicar, mediante flujogramas, el funcionamiento del programa.

Nociones astrométricas.

Para comprender mejor que esperamos de esta librería nos vendrá bien dar unas breves nociones sobre astrometría. Empezaremos explicando como se define la posición de un objeto en el cielo.

Para ello tendremos que introducir el concepto de esfera celeste. La esfera celeste es una esfera imaginaria de radio indefinido y concéntrica al globo terreste.

El ojo humano, al no poder percibir las diferentes distancias de los objetos celestes, creerá que estos se encuentran en el mismo plano. Esto es, están colocados en la esfera anteriormente mencionada.

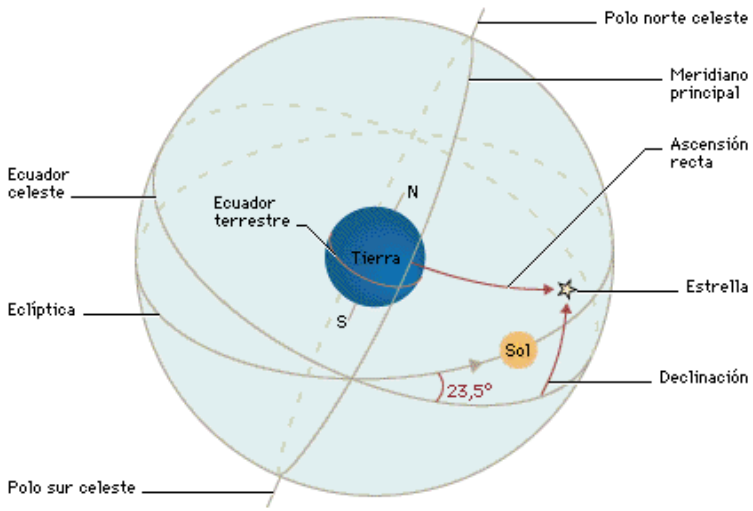


Figura 68. Esfera celeste (Masm, 2004).

Esta pequeña mentira nos permitirá definir las posiciones de estos objetos como si estuvieramos trabajando con un globo terráqueo.

Nuestra esfera celeste, al igual que la terrestre, tendrá un ecuador y dos polos sobre los que fijar las coordenadas. Así la posición de un objeto en la esfera se medirá mediante su ascensión recta y su declinación.

En nuestro proyecto nos interesará medir la posición del satélite utilizando su altura y su acimut. Utilizando la misma esfera celeste estas coordenadas estarán referidas al horizonte local del observador. Es por eso que nuestro script necesitará de la posición del observador sobre el globo terraqueo para poder funcionar.

La altura corresponderá con la distancia angular entre la estrella y el horizonte local. El acimut, por el contrario, se calculará midiendo el ángulo existente entre la proyección vertical del satélite con el horizonte local y el norte celeste.

La siguiente imagen nos dará una idea más clara de estos dos conceptos.

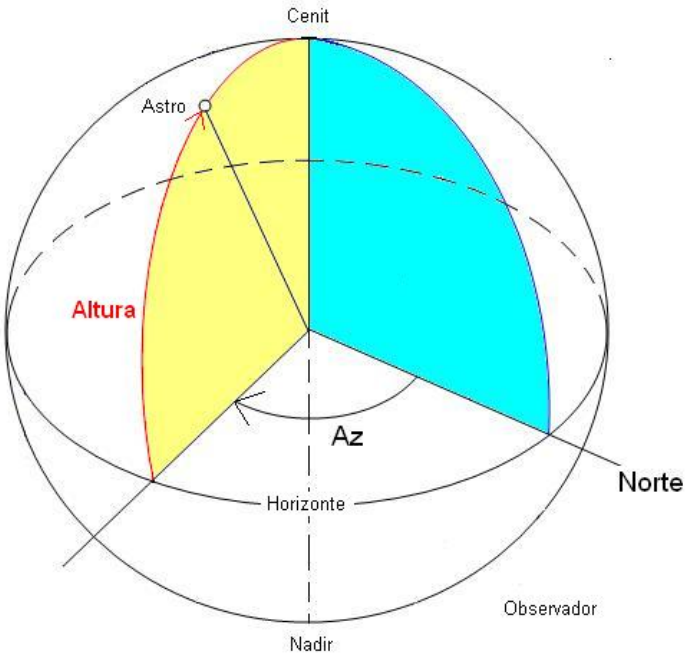


Figura 69. Altura y acimut (AZ) de un astro cualquiera (Elias, 2007)[140].

Podemos denominar a la altura y acimut de un objeto como las coordenadas horizontales de este.

[140] http://es.wikipedia.org/wiki/Altura_%28astronom%C3%ADa%29

Obtención de las coordenadas horizontales del satélite.

Para obtener estos dos valores de la librería de cálculo tendremos que crear un objeto cuyos atributos serán las condiciones particulares de nuestro observador.

En el archivo `calculo.py` nos encargamos de este proceso. Para ello definimos una clase llamada `Calculo` que, pasándole como argumentos los valores de nuestro observador, nos devuelve los valores actuales del objeto seleccionado.
A continuación desgranaremos el código fuente de dicho archivo.

Lo primero será crear la clase. Siguiendo los modismos de este lenguaje de programación el nombre de la clase tendrá su primera letra, la 'c', en mayúscula.

```
1  class Calculo():
```

Definiremos la función constructora de la clase creando todas la variables necesarias.
Para que podamos modificar el valor de estas a nuestro conveniencia estas variables tendrán valores nulos.

```
2      def __init__ (self, timetuple = None, dia = None, mes = None,
anio = None, hora = None, minuto = None, segundo = None, observador =
None, longitud = None, latitud = None, elevacion = None, horizonte =
None, presion = None, satelite = None, tle = None, altura = None,
acimut = None, ascensionrecta = None, declinacion = None):
```

Importamos los módulos necesarios para el funcionamiento de este script.
Estos son: `time`, para la obtención del tiempo, `ephem`, nuestro módulo de cálculo de posiciones y `os`, necesario para poder trabajar con los directorios del sistema.

```
3            import time
4            import ephem
5            import os
```

Crearemos un objeto de la clase `time()`.

```
6            tiempo = time.time()
```

Mediante herencia crearemos un objeto de la clase `gmtime()` basándonos en el objeto anteriormente creado.

```
7            a = time.gmtime(tiempo)
```

Los atributos del anterior objeto serán los valores actuales de la fecha y hora del sistema. Tendremos que crear una tupla con ellos.

```
8            b = a.tm_year, a.tm_mon, a.tm_mday, a.tm_hour, a.tm_min,
a.tm_sec
```

Tanto la tupla de datos como los valores concretos de la fecha y la hora serán asignados a nuevas variables . Como podremos comprobar todas y cada una de las nuevas variables están precedidas por la declaración "self". Esto nos permitirá acceder a estos valores desde fuera del objeto.

```
9           self.timetuple = b
10          self.dia = a.tm_mday
11          self.mes = a.tm_mon
12          self.anio = a.tm_year
13          self.hora = a.tm_hour
14          self.minuto = a.tm_min
15          self.segundo = a.tm_sec
```

La posición del satélite en el fichero de su familia viene dada por un índice. Ya que este número viene con un tipo de datos string tendremos que convertirlo a un número entero para poder trabajar con él.

```
16          satelite = int(satelite)
```

Con el anterior número podremos crear nuevos índices para las tres filas de texto necesarias para definir un objeto. Asignaremos a cada índice un nombre que nos permita identificarlo con facilidad.

```
17          titulo_TLE = satelite*3
18          indice_linea1 = titulo_TLE + 1
19          indice_linea2 = titulo_TLE + 2
```

Los valores de elevación, presión y horizonte del observador serán transformados a variables de tipo flotante.

```
20          elevacion = float(elevacion)
21          presion = float(presion)
22          horizonte = float(horizonte)
```

Con todos los datos necesarios para poder trabajar ya podremos crear el objeto que defina a nuestro observador. Necesitaremos un objeto de la clase `Observer()`.

```
23          observador = ephem.Observer()
```

Asignaremos a este nuevo objeto nuestros atributos. Estos serán longitud, latitud, elevación, presión, horizonte y hora actual.

```
24          observador.lon = longitud
25          observador.lat = latitud
26          observador.elevation = elevacion
27          observador.pressure = presion
28          observador.horizon = horizonte
29          observador.date = ephem.Date(self.timetuple)
```

Todavía nos faltan los valores orbitales del satélite. Para obtener estos tendremos que buscar el archivo de la familia correspondiente y dar con las líneas necesarias dentro de este.
El primer paso será acceder al directorio que contiene los elementos orbitales. Antes de eso guardaremos el directorio actual del sistema para poder regresar a este una vez terminadas las lecturas necesarias

```
30          directorio_trabajo = os.getcwd()
```

Ahora podremos desplazarnos a la carpeta que contiene los elementos orbitales.

```
31          os.chdir(directorio_trabajo + '/TLEs')
```

El nombre del archivo viene definido por la variable tle. El valor de tle es asignado al objeto en el momento de su creación. Abriremos el archivo en modo de lectura.

```
32          lectura_TLEs = open(tle, 'r')
```

Leeremos cada línea del archivo. Cada línea se guardará en un fichero utilizando el formato de lista siendo cada línea un elemento de esta. También tendremos que eliminar los saltos de línea existentes al final de cada una de ellas.

```
33          lista_TLEs = lectura_TLEs.readlines()
34          lista_TLEs = [item.rstrip('\n') for item in lista_TLEs]
```

Crearemos las tres variables necesarias para la función readtle perteneciente al módulo ephem. Estas corresponden al nombre del objeto y a las dos líneas de datos orbitales.
Usaremos los índices creados anteriormente.

```
35          objeto = lista_TLEs[titulo_TLE]
36          linea1 = lista_TLEs[indice_linea1]
37          linea2 = lista_TLEs[indice_linea2]
```

Le pasaremos las variables creadas a la función readtle creando un nuevo objeto.

```
38          v = ephem.readtle(objeto, linea1, linea2)
```

Con la función compute obtendremos todos los valores deseados de nuestro satélite.

```
39          v.compute(observador)
```

Antes de asignar las coordenadas calculadas a nuevas variables tendremos que volver al directorio principal del script.

```
40          os.chdir(directorio_trabajo)
```

Los datos obtenidos serán atributos públicos de nuestro objeto “v”. Como queremos acceder a ellos desde fuera de nuestro objeto asignaremos estos atributos a nuevas variables definidas como atributos públicos de nuestra clase Calculo().

```
41          self.altura = v.alt
42          self.acimut = v.az
43          self.ascensionrecta = v.ra
44          self.declinacion = v.dec
45          self.orto = 'orto'
```

Este será el proceso de cálculo de las coordenadas del satélite escogido. Cada vez que deseemos obtener estos datos tendremos que realizar todos los pasos mencionados anteriormente.

Las posibles mejoras del código descrito se mencionarán en el último capítulo de esta memoria.

Flujograma del programa.

Para tener una visión más clara del funcionamiento de este desarrollo realizaremos un flujograma de cada proceso del sistema.

Así podremos dividir el comportamiento del script en los siguientes apartados:

- Respuesta de cada una de las ventanas del programa a las acciones del usuario.
- Descarga manual y automática de los elementos orbitales del sitio web Celestrak.
- Protocolo de comunicaciones entre el controlador y el sistema de rotores mediante el puerto serio de ambos dispositivos.
- Diferentes clases creadas para realizar tareas concretas. Entraremos en su definición más adelante.

Comportamiento de la interfaz.

Como mencionanos en el capítulo anterior el programa consta de una ventana principal, que consiste en un objeto de la clase Tkinter y tres ventanas secundarias anidadas a esta.

Analizaremos el comportamiento de cada ventana. Para ello describiremos su creación y su funcionamiento una vez estén operativas.

Ventana principal.

Esta ventana funcionará mediante un loop infinito. El código de la clase Ventana_principal() será ejecutado una y otra vez hasta que salgamos de este destruyendo la ventana generada.

Este comportamiento se puede resumir en el siguiente diagrama.

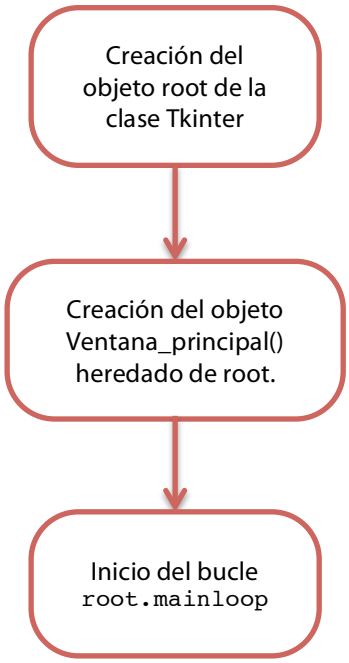


Figura 70. Flujograma simplificado del fichero main.py.

Bucle principal.

El constructor de la clase Ventana_principal realizará las siguientes acciones.

- Creación de la tipografía del sistema. Para facilitar la modificación de los tipos de letra de la interfaz estos se crean mediante una llamada a una función en el constructor de la clase.
- Creación de los widgets. Una única función se encarga de recoger la creación de los widgets del sistema. Se incluirán métodos para depurar posibles errores del código además de sentencias para seleccionar distinto código dependiendo del sistema operativo utilizado para ejecutar este programa.
- Puesta en marcha de la hora. Se creará un proceso independiente para mostrar la hora del sistema. Para evitar conflictos con otros procesos este funcionará como un demonio.
- Colocación de un mensaje de advertencia. El script muestra en la ventana principal del programa diferentes mensajes de advertencia. En este punto se colocará el primer mensaje.

Podremos resumir el texto anterior en el siguiente flujograma.

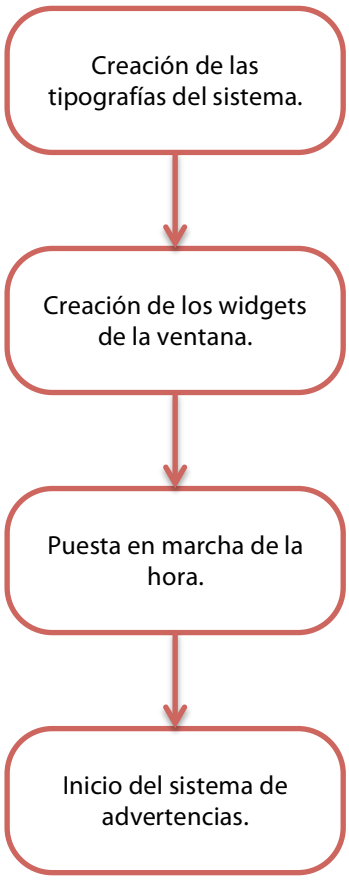


Figura 71. Flujograma simplificado del constructor de la clase Ventana_principal.

Listado de localizaciones.

Esta ventana consistirá en un objeto de la clase Toplevel() perteneciente al módulo Tkinter. Este objeto se comportará como una ventana anidada de nuestra ventana principal. Esto significa que si cerramos la ventana principal está también se cerrará y solo podremos acceder a nuestra interfaz anidada a través de la interfaz principal.

El código fuente de esta ventana nos permitirá gestionar nuestra pequeña base de datos de localizaciones.
Esta base de datos, que se encuenta en un formato csv, será poco eficiente para gestionar grandes volumenes de información. Esto no nos causará ningún problema ya que suponemos que los registros nunca serán muy numerosos.
También accesible mediante un editor de texto plano nuestra base de datos se encontrará en el fichero "localizaciones.list".

Flujograma del sistema de gestión de localizaciones

El siguiente flujograma nos mostrará el comportamiento de esta ventana.

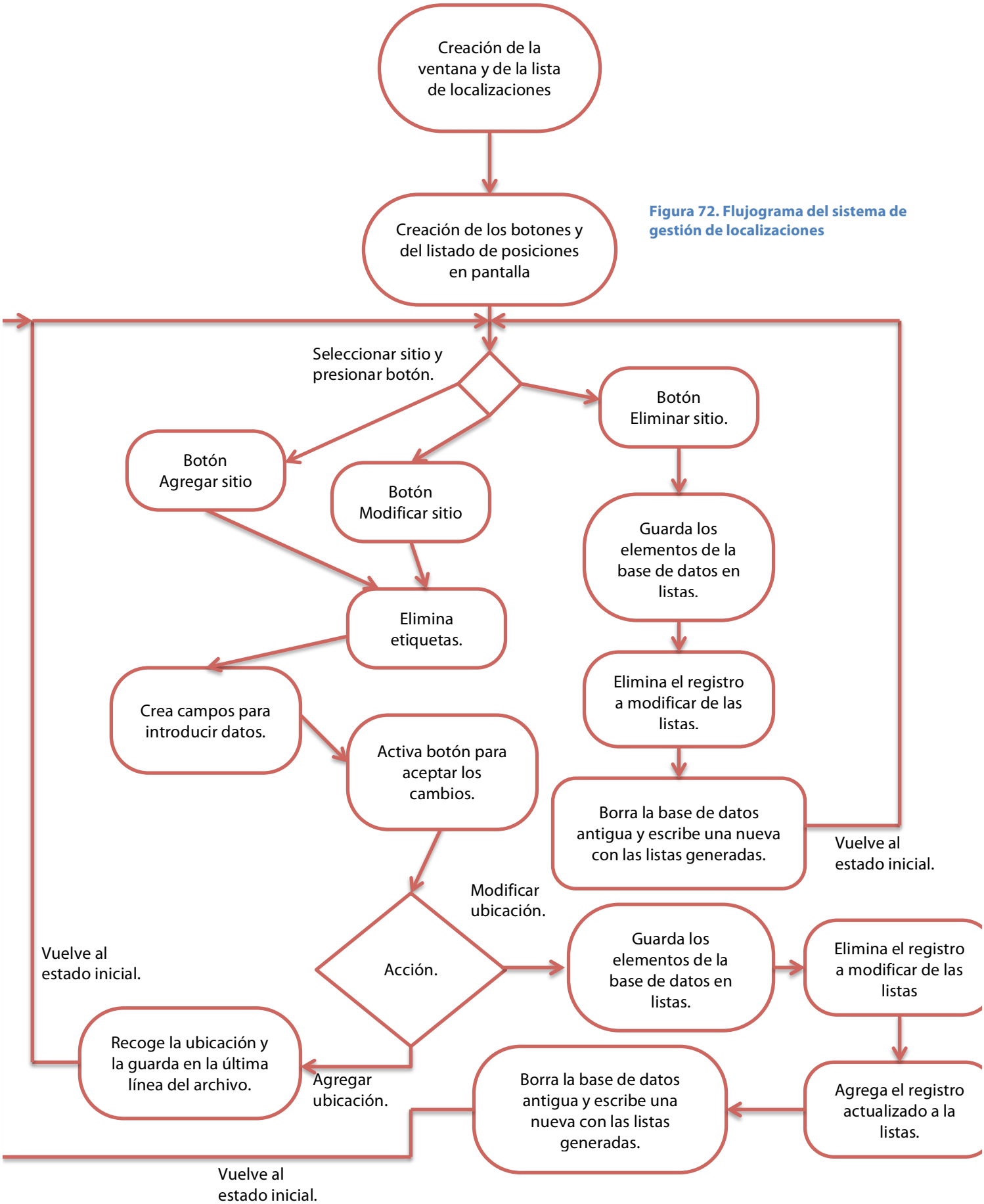


Figura 72. Flujograma del sistema de gestión de localizaciones

Ventana de configuración.

Esta ventana consta de dos funcionalidades principales. Como mencionamos anteriormente, la ventana de configuración nos valdrá para gestionar, además de nuestra librería de elementos orbitales, nuestas configuraciones de conexión personalizas.

En este caso realizaremos dos flujogramas, uno para cada acción mencionada.

Flujograma del sistema de vista previa de ficheros.

Para llevar un mejor control de las librerías presentes en el sistema el script nos permitirá, pulsando sobre el nombre de una librería, comprobar que satélites se encuentran presentes en ella.

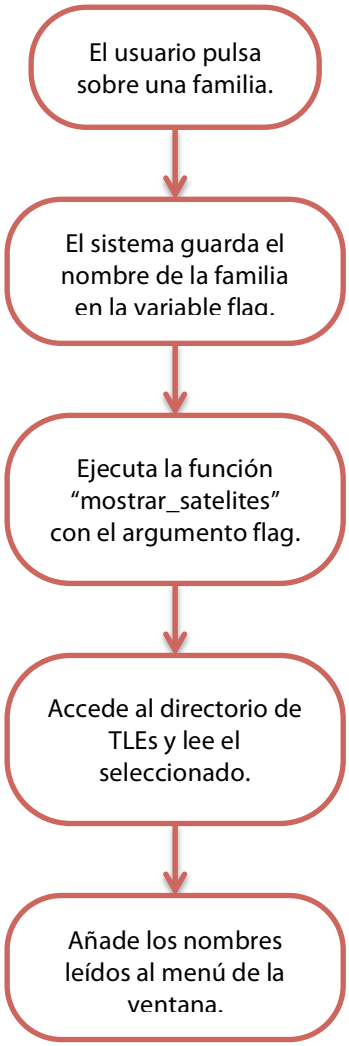


Figura 73. Diagrama de flujo del sistema de vista previa de ficheros.

Flujograma del sistema de almacenamiento de configuraciones.

El programa permitirá almacenar en un fichero de texto las configuraciones definidas por el usuario. Estas estarán referidas al protocolo de conexión de nuestro dispositivo con el sistema de rotores.

Podremos definir los siguientes elementos:

- Nombre de la configuración.
- Velocidad de transferencia.
- Longitud del byte.
- Paridad.
- Bits de parada.
- Puerto serie utilizado.

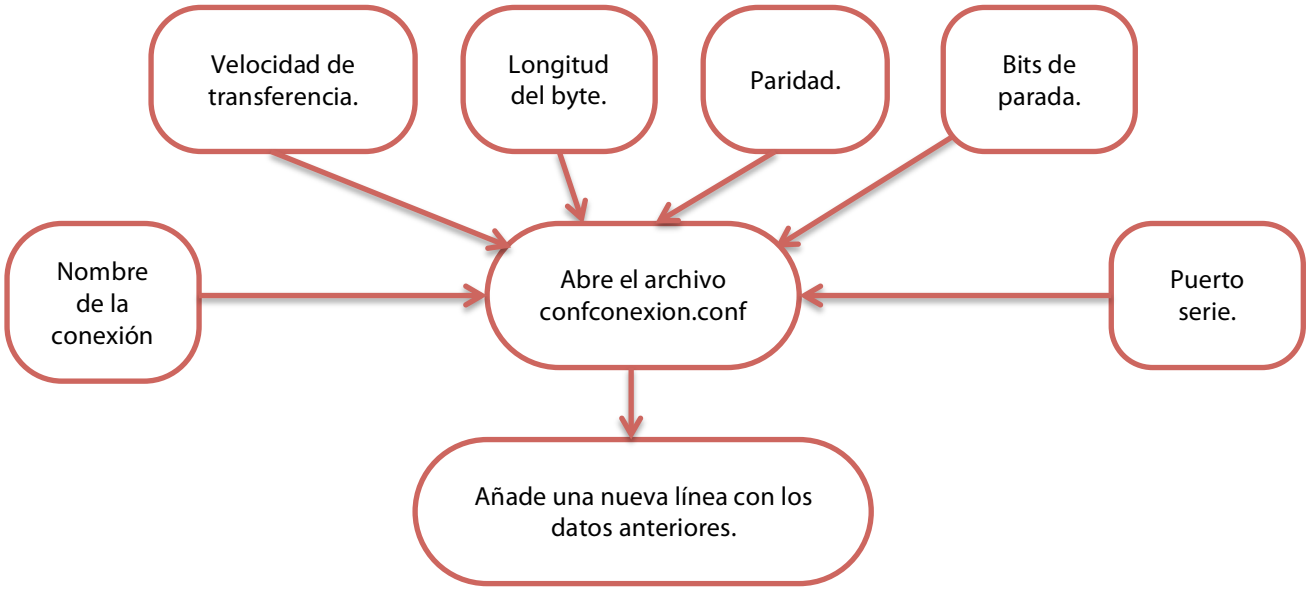


Figura 74. Recolección de parámetros de conexión para la base de configuraciones del sistema.

Descarga de elementos orbitales.

En una primera medida se ha recurrido al módulo wget para que el script descargue los elementos orbitales.

Instalación del módulo de descarga de ficheros.

Para instalar este módulo acudiremos a la utilidad pip. Esta utilidad, que aparece en el cuarto capítulo de esta memoria, nos permitirá instalar wget y, además, mantener una versión actualizada del mismo.
Haremos uso del método correspondiente al sistema operativo que estemos utilizando, estos también están detallados en el capítulo mencionado anteriormente

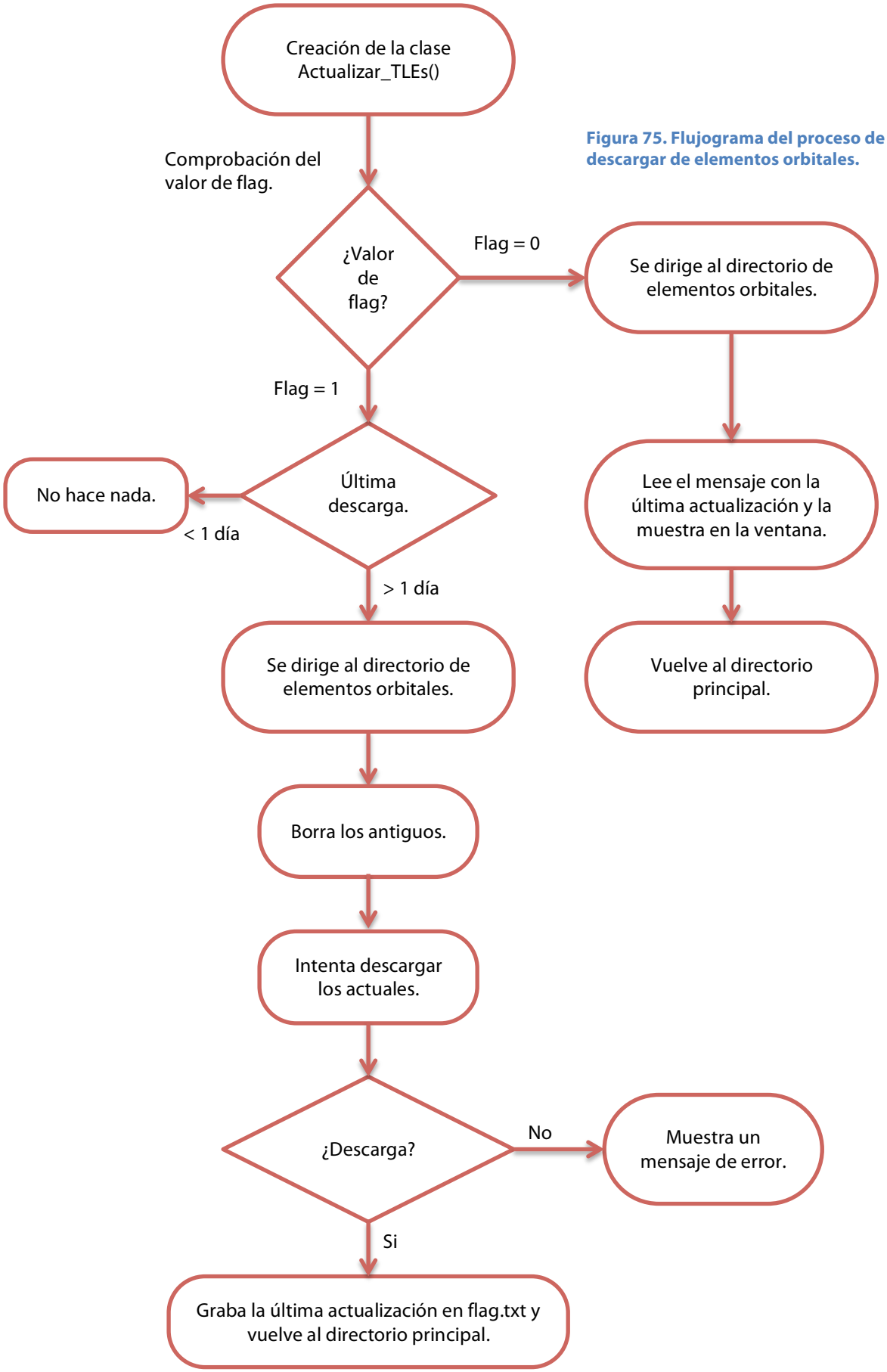


Figura 75. Flujograma del proceso de descargar de elementos orbitales.

Protocolo de comunicaciones.

Para realizar las conexiones seriales del sistema utilizaremos el módulo PySerial (Liechti, 2013)[141].
En la web del creador podremos encontrar gran cantidad de documentación sobre este desarrollo.

No entraremos en la definición de las ordenes ni el protocolo a usar hasta el siguiente capítulo. En él, al realizar las pruebas de funcionamiento del sistema, iremos informando de que comandos hemos enviado al rotor desde nuestro sistema.

[141] http://pyserial.sourceforge.net/

7 – Interfaz de conexión.

En este capítulo nos dedicaremos a construir y ensamblar cada una de los elementos que integran este prototipo.

En primer lugar montaremos los componentes de la interfaz de conexión realizando a continuación la programación del microcontrolador de esta. También tendremos que preparar un cable para conectar esta placa al controlador del rotor del que disponemos.

Conectaremos nuestra placa de prototipado con la interfaz de conexión mediante el convertidor de nivel de tensión MAX3232. Realizaremos la pruebas necesarias para comprobar el buen funcionamiento de este.

Por último lugar montaremos todos los dispositivos en un caja adecuada para evitar daños.

Construcción de la inferfaz de conexión

Este punto consta de dos pasos. En primer lugar soldaremos los componentes en la placa. La lista de estos se encuentra en la web y se pueden obtener en cualquier tienda de electrónica.

Una vez preparada la placa tendremos que programar el microcontrolador de esta.
La placa está diseñada para no necesitar un programador externo así que solo tendremos que conectarla a un ordenador y ejecutar el programa proporcionado por el creador del proyecto para grabar el integrado.

A la facilidad de no tener que usar otro dispositivo para grabar el microcontrolador se contrapone el hecho de que el software que ofrece el desarrollador solo está disponible para plataformas de la familia Windows.

El programa y el código fuente se encuentra en el cd que acompaña a esta memoria.

Soldado de los componentes.

Soldaremos los componentes del dispositivo siguiendo el orden recomendado por el creador del sistema. Este, presente en el manual disponible en la web del desarrollador, es el siguiente:

1. Resistencias. A excepción de R9, R10, R11 y R12 que no son necesarias.
2. Condensadores cerámicos.
3. Condensadores electrolíticos.

4. Potenciometro de ajuste.
5. Zócalos para los circuitos integrados.
6. Pines.
7. Conector de nueve pines para el puerto serie.
8. Zócalo para la pantalla de cristal liquido. Opcional.
9. Diodos.
10. Transistores.
11. Interruptor de reinicio.
12. Cristal oscilador.
13. Integrado 7805.

En el proceso de soldado de la placa nos hemos encontrado con varios problemas. Pasaremos a detallar cada uno de ellos y trataremos de explicar lo mejor posible como los afrontamos.

Resistencias.

El hecho de utilizar una placa ya preparada tiene la enorme desventaja de que si no se dispone del componente correcto podemos tener complicaciones con el espacio disponible.

El primer problema en el ensamblaje de la placa vino motivado por el tipo de resistencias utilizadas por el creador del dispositivo. Este, por temas de espacio, utilizó resistencias con una potencia de disipación de un octavo de vatio.
El primer atolladero fue que estas son dificiles de encontrar en tiendas de venta al por menor.
Para ello hubo que utilizar resistencias de un cuarto de vatio montándolas verticalmente en la placa.

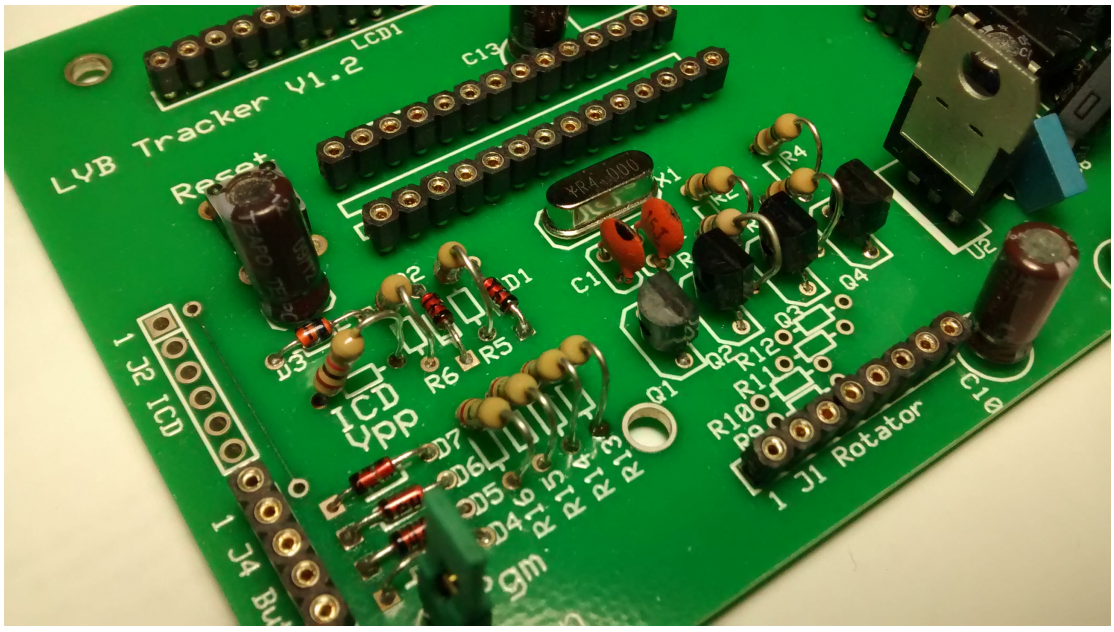


Figura 76. Detalle del montaje de las resistencias en la placa.

Potenciómetro de ajuste.

El dar con el modelo de potenciómetro sugerido también trajo problemas.
La disposición de los pines y el tamaño de estos son distintos en el modelo propuesto y el existente en las tiendas de la zona.

Hubo que doblar y ajustar los pines para que entraran en sus orificios correspondientes.

Esta solución, aunque poco segura, se antoja valida para realizar las pruebas de funcionamiento.

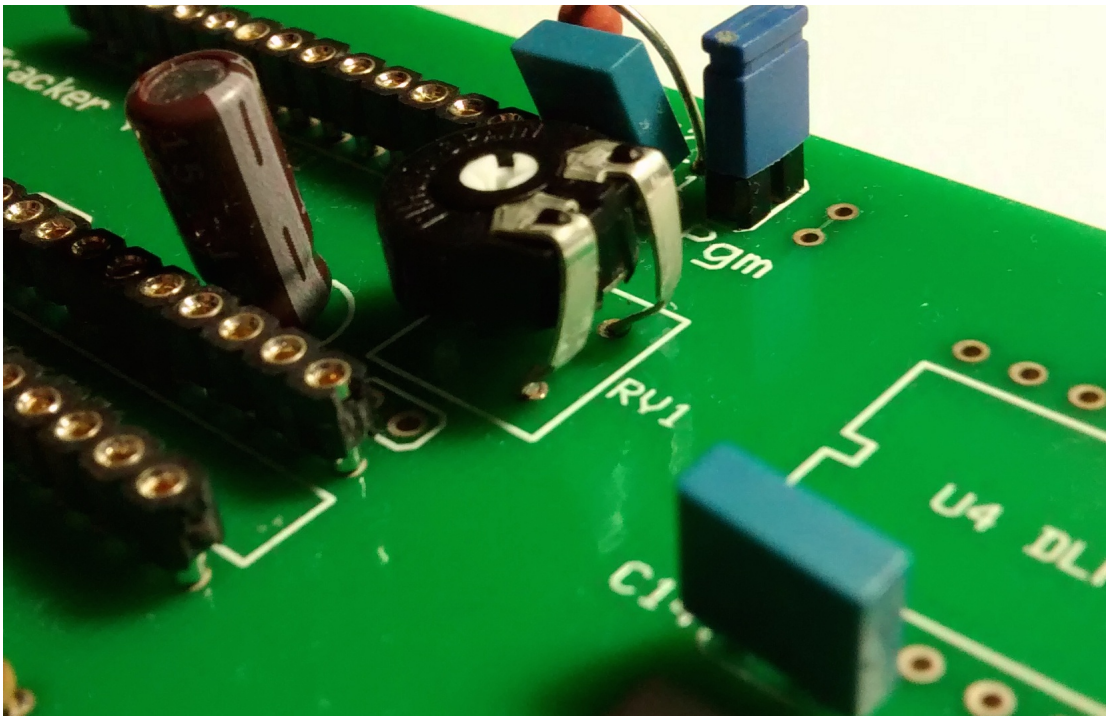


Figura 77. Detalle del soldado de los pines del potenciómetro a la placa.

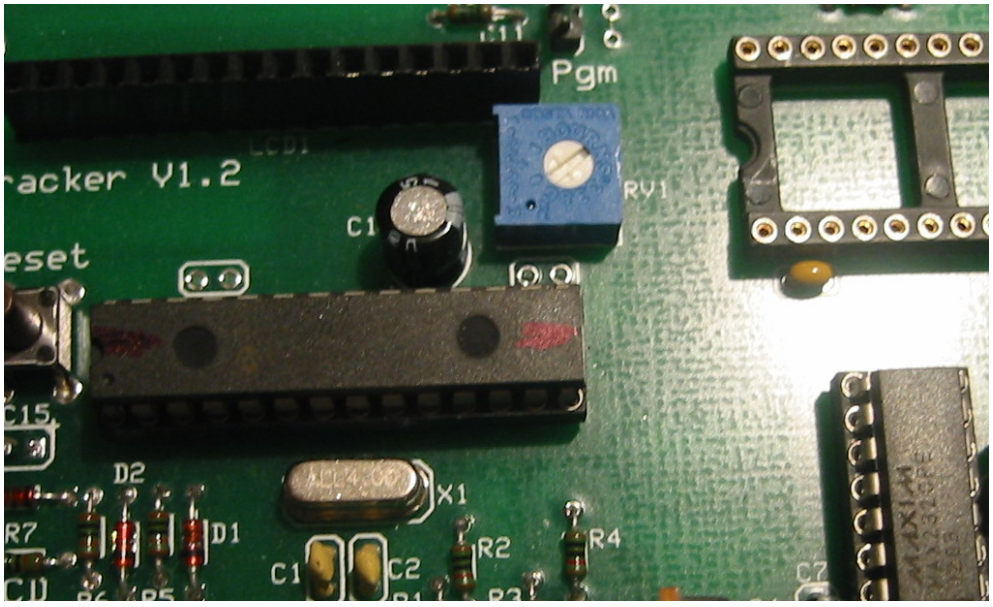


Figura 78. Montaje con el potenciómetro recomendado.

Podemos comparar nuestro montaje con la fotografía del montaje del potenciómetro del creador de la interfaz.

Se palpa la necesidad de, en futuros prototipos, realizar nosotros mismos la placa para así no tener conflictos con los componentes.

Regulador 7805.

El hecho de tener que utilizar componentes de un tamaño superior a los originales redució el espacio disponible para el disipador del regulador 7805.
Para tiempos cortos de funcionamiento dejaremos el regulador sin disipación pero para prototipos futuros, destinados a estar siempre encendidos, modificaremos el layout de la placa para dejar sitio a un disipador.

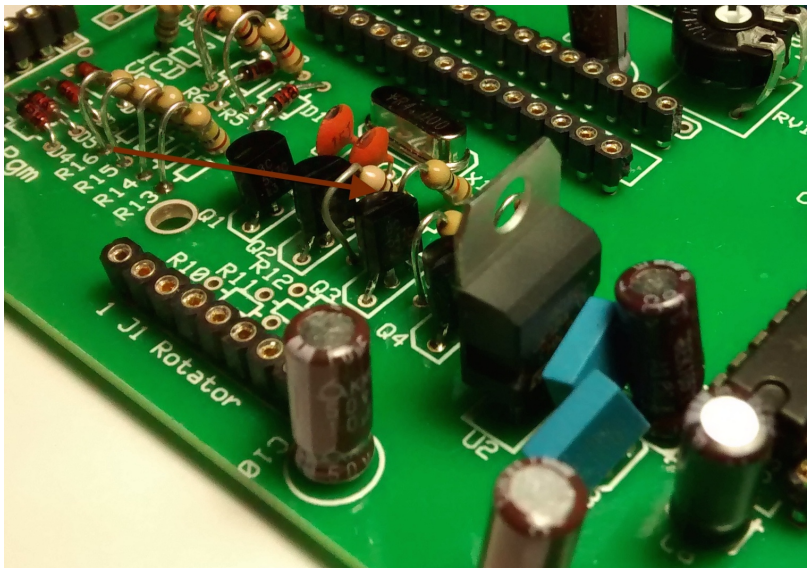


Figura 79. Fotografía de la zona del regulador.

Hemos señalado con la flecha roja la posición donde debería encontrarse el disipador. Como podemos ver el disipador choca con el transistor Q4 y con la resistencia C10.

Soldadura final.

Aunque la calidad de las soldaduras es un tanto irregular, debido a la inexperiencia del que escribe estas líneas, el resultado final parece lo suficientemente correcto para su funcionamiento.

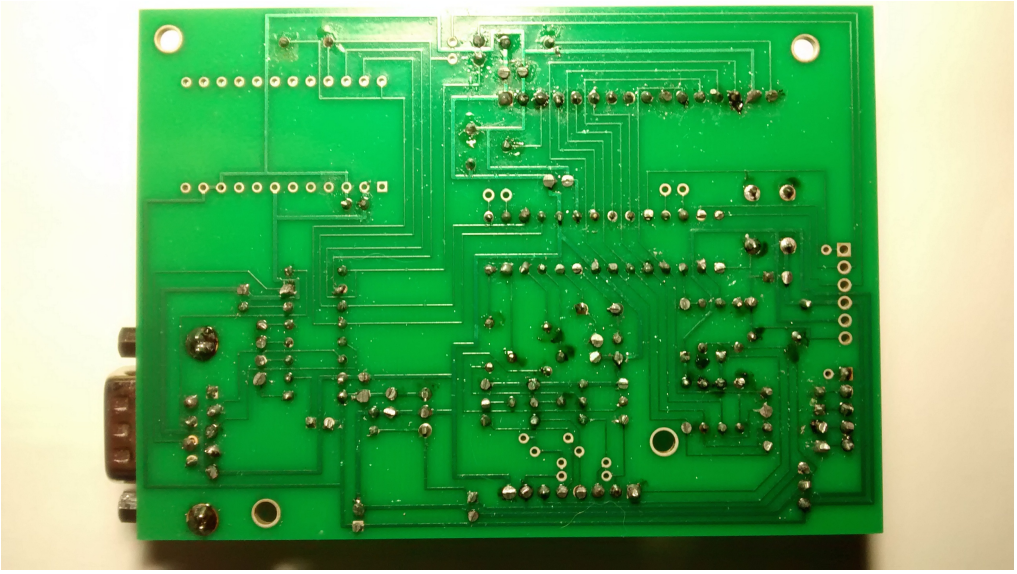


Figura 80. Cara de las soldaduras de la placa.

Programación del microcontrolador.

En este apartado encontraremos otro problema y es que la energía de alimentación de la placa viene de los rotores. Así, hasta que no realicemos las pruebas in situ del sistema, no podremos grabar el código en el microcontrolador.

En el próximo capítulo retomaremos este tema.

Conexión de la placa de prototipado a la interfaz de conexión.

Conversor UART-TTL a RS232

En este punto es cuando necesitaremos un dispositivo que comunique nuestra interfaz de conexión, LVB Tracker, con nuestro sistema de control, Raspberry Pi.

Para ello, como comentamos en la página 70, utilizaremos un integrado MAX3232.
Para ahorrar tiempo y dinero compraremos una placa ya creada que posea el anterior dispositivo.

Estas se encuentran disponibles en sitios de venta online tan populares como eBay con precios a partir de los cinco euros.

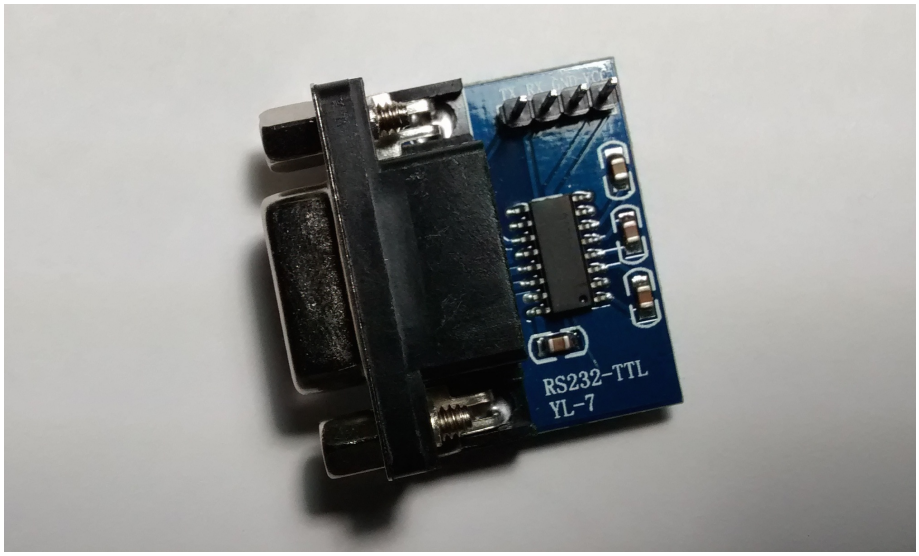


Figura 81. Conversor de niveles de tensión TTL a RS232.

En la fotografía superior podemos ver la placa con el integrado MAX3232, los cuatro pines de conexión y los cuatro condensadores. Todos ellos componentes con un montaje superficial.

Pruebas de funcionamiento del conversor de niveles de tensión.

Antes de montar este dispositivo en la versión final de nuestro prototipo tendremos que realizar algunas pruebas de funcionamiento. En ellas conectaremos nuestra Raspberry Pi al ordenador e intentaremos transmitir cadenas de texto entre ambos dispositivos.

Esquema de conexión.

Figura 82. Convertidor comercial de señal.

Ya que el portátil que estamos usando, un MacBook White del año 2009, no dispone de un puerto RS232 tendremos que utilizar un conversor para disponer de dicho puerto.

Para ello usaremos un convertidor de niveles de tensión USB a RS232. Estos están disponibles en cualquier tienda de informática con precios que no sobrepasan los diez euros.

Yendonos al otro extremo de la conexión, nuestra Raspberry Pi, tendremos que conectar el convertidor TTL a RS232 a cuatro pines de la placa.

El convertidor no implementa las líneas necesarias para la comprobación de la conexión, así que los puertos que usaremos solo serán los siguientes:

- Tensión. Al trabajar con un MAX3232 solo necesitaremos 3'3 voltios.
- Tierra. Nos valdrá cualquier tierra de la placa.
- TD. Envío de la señal.
- RD. Recepción de la señal.

Según la wiki (eLinux.org, 2014)[142] oficial en elinux.org estos puertos corresponderán a los siguientes pines:

- Tensión. Pin número 1 y 17. Usaremos el número 1 por razones prácticas.
- Tierra. Pin número 6, 9, 14, 20 y 25. Por comodidad usaremos el puerto número 6.
- TD. Pin número 8. También puede ver indicado por la denominación GPIO14.
- RD. Pin número 10. Se puede encontrar también como GPIO15.

Quedando el cableado de la manera que podemos ver en la siguiente fotografía.

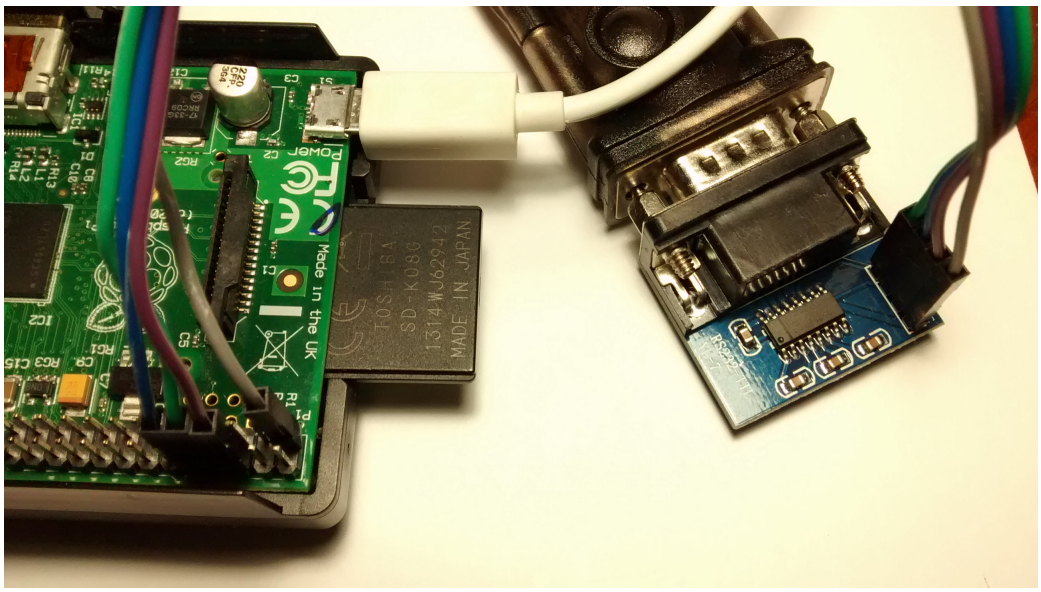


Figura 83. Adaptador TTL-RS232 y Raspberry Pi conectados.

[142] http://elinux.org/RPi_Low-level_peripherals#General_Purpose_Input.2FOutput_.28GPIO.29

Resumiendo, la conexión entre el ordenador y la Raspberry Pi quedará de la siguiente manera.

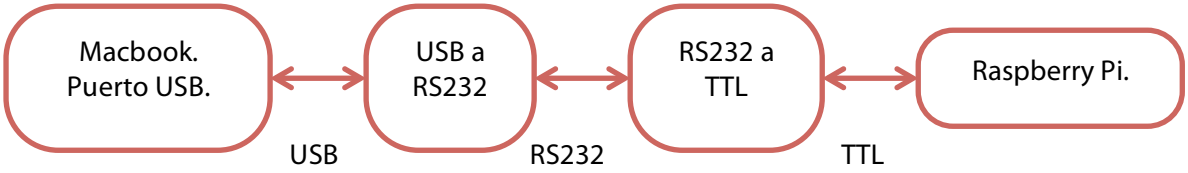


Figura 84. Esquema de conexión.

Software utilizado.

Para realizar las pruebas utilizaremos diferentes programas para cada sistema operaivo.
En Mac OS X, después de probar con alternativas como QuickTerm, utilizaremos un pequeño programa perteneciente a la librería pySerial.

Para ejecutarlo tendremos que teclear en la línea de comandos el siguiente código.

```
1  python -m serial.tools.miniterm
```

El anterior comando nos dará la siguiente salida por pantalla.

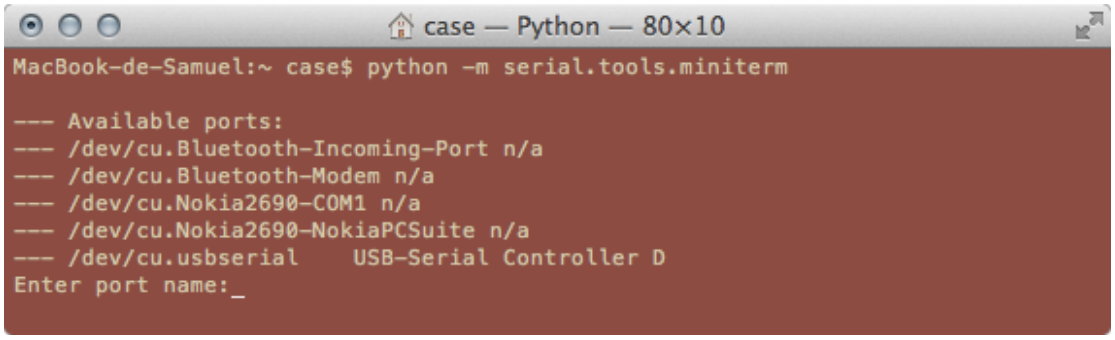


Figura 85. Salida por pantalla.

En nuestro caso el puerto correcto es el último de la lista. Para establecer la conexión solo tendremos que introducir la dirección de este.

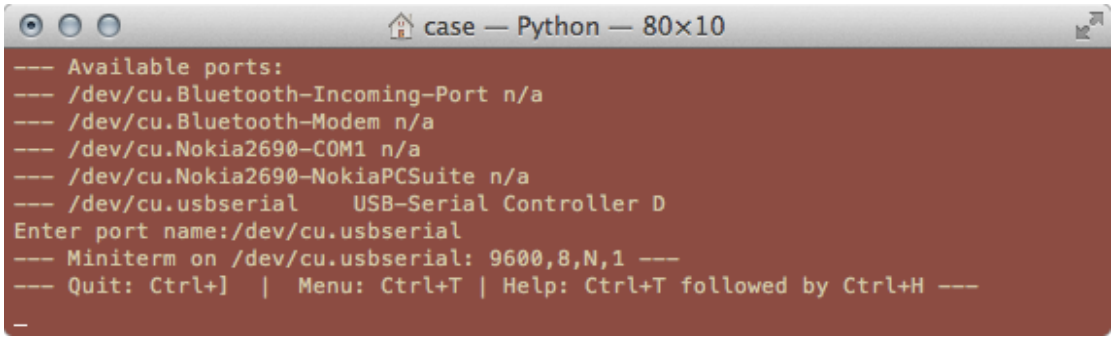


Figura 86. Conexión establecida entre nuestro Macbook y nuestra Raspberry Pi.

Por defecto creará una conexión de 9600 baudios con 8 bits por byte sin paridad y con un bit de parada.
Ahora, el texto que envíemos por esta ventana se transmitirá por el puerto seleccionado.

En nuestra Raspberry Pi utilizaremos minicom. Este en un programa destinado al establecimiento de conexiones seriales mediante la línea de comandos.
Para crear una conexión similar a la anterior tendremos que ejecutar el siguiente comando:

```
1  minicom -b 9600 –D /dev/ttyAMA0
```

Ejecutado esta orden obtendremos una ventana donde se mostrará el texto recibido y, además, podremos escribir el texto que queramos enviar.

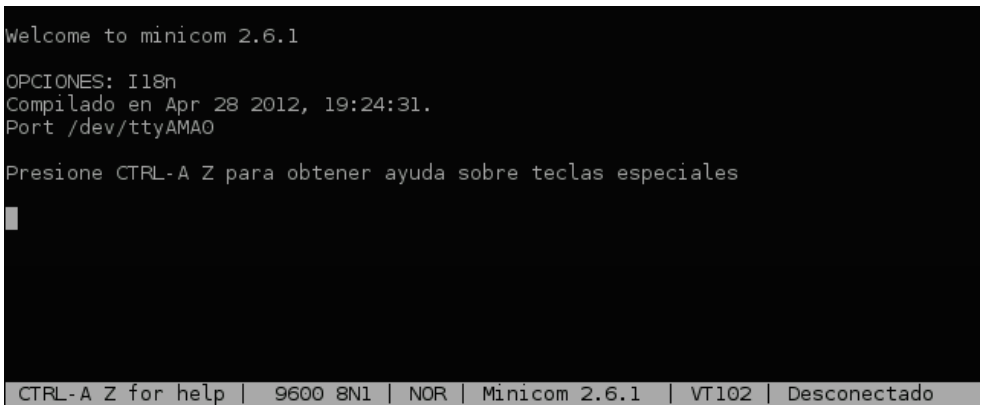


Figura 87. Conexión establecida en la Raspberry Pi hacia nuestro MacBook.

Por defecto las conexiones creadas tendrán las siguientes características:

- 8 bits por byte.
- Sin paridad.
- 1 bit de parada.

A continuación probaremos el sistema creado.

Envío de mensajes.

Una vez establecida la conexión envíaremos el siguiente mensaje desde Mac OS X.

```
Texto de prueba.
```

Este texto nos devolverá la siguiente salida en Raspbian.

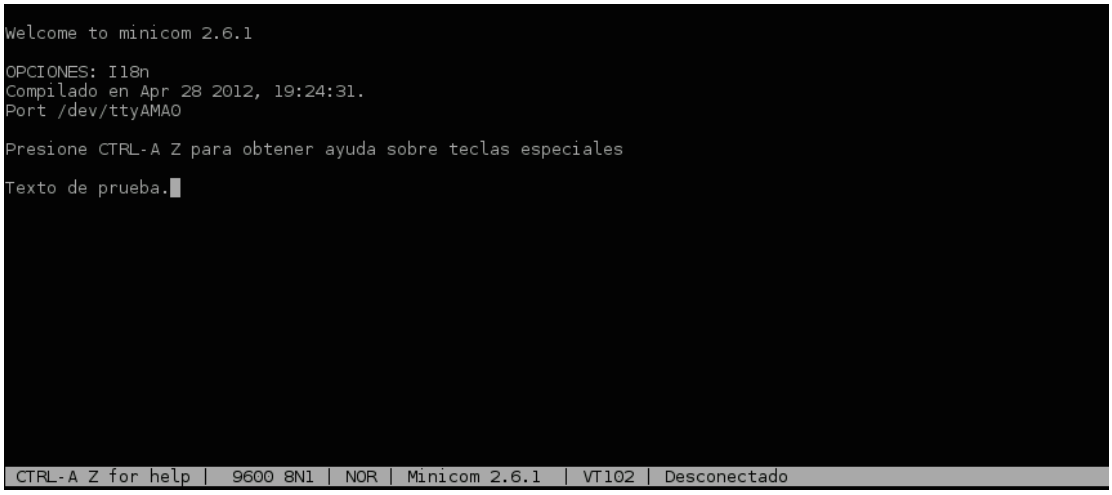


Figura 88. Salida por pantalla en Raspbian.

El mismo mensaje escrito en Linux nos proporcionará la siguiente salida en Mac OS X.

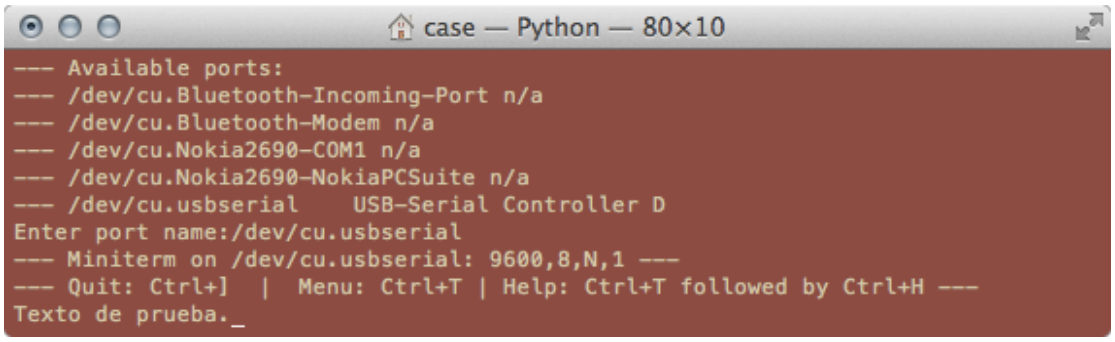


Figura 89. Salida por pantalla en Mac 10.9.3.

Como podemos ver el funcionamiento de los convertidores es correcto. No deberían darnos ningún problema cuando se encuentren activos.
Pasamos al siguiente capítulo, Conclusiones, donde realizaremos las pruebas prácticas in situ.

8 – Conclusiones.

Este capítulo se dividirá en dos secciones principales.

En la primera de ellas abordaremos las pruebas del sistema anteriormente desarrollado. Estas se realizarán utilizando el radiotelescopio instalado en la azotea del edificio de i+d+i de la Universidad Politécnica de Cartagena.
Trataremos de poner al sistema en la peor situación posible para así asegurarnos de su buen desempeño.

La segunda parte de este capítulo consistirá en el análisis de los datos recabados.
Para ello tendremos en cuenta factores tales como la facilidad de uso del sistema, la fiabilidad de éste y la precisión del mismo.
Las conclusiones a las que lleguemos nos ayudarán a, en el próximo capítulo, enumerar las posibles mejoras del prototipo.

Conexión inicial del sistema.

En este punto conectaremos todos los elementos del radiotelescopio y, mediante un software comercial, comprobaremos el buen funcionamiento del mismo.

Esto es vital para la comprobación de nuestro prototipo, tenemos que asegurarnos que los problemas que nos encontraremos en el desarrollo del mismo solo serán de nuestro sistema.

Para las pruebas utilizaremos el software de guiado Orbitron© (Stoff, 2007)[143]. Solo disponible para sistemas Windows, instalaremos este programa en nuestro portátil de pruebas, un Hewlett-Packard xe4100.

Desde el sitio web oficial del desarrollo podremos obtener este desarrollo. En él encontraremos guías de uso para su instalación y configuración.

Estado inicial de los componentes.

Antes de realizar cualquier prueba documentaremos el estado inicial de los distintos componentes del sistema.
A continuación vienen una serie de fotografías realizadas in situ del receptor y del radiotelescopio.

Radiotelescopio.

El estado del radiotelescopio, tras varios años de inactividad, se nos antoja un tanto deficiente.

[143] http://www.stoff.pl/

El hecho de encontrarse a la intemperie y sin mantenimiento ha propiciado que el conjunto se encuentre en un aparente mal estado.

Como podemos observar en la foto se antoja necesario un mantenimiento de la zona para cumplir con unas medidas básicas de limpieza.

Figura 90. Base del radiotelescopio.

Los motores han sufrido los estragos del tiempo y la falta de cuidados periódicos.
Además de la corrosión observable a siempre vista podemos comprobar como estos se encuentran cubiertos por deposiciones de aves.

Figura 91. Rotores del radiotelescopio.

Una vez comprobado el estado de los motores pasamos a revisar el estado de los cables.

Estos, que se encuentran junto al radiotelescopio, han corrido una suerte similar al anterior.

A simple vista podemos cerciorarnos que los conectores también se encuentran afectados por la corrosión.

Figura 92. Conector de cuatro pines del radiotelescopio.

Controlador de los rotores.

Almacenado en los sotanos del edificio de Antiguones el estado del controlador de los rotores es perfecto. Suponemos que no nos dará ningún problema.

Figura 93. Controlador de rotores.

La conexión de los rotores al controlador se realizará mediante dos cables de 4 hilos.
Cada cable se encargará del control de un motor. Estos se conectarán en la parte trasera del dispositivo.

Figura 94. Parte trasera del controlador.

También encontramos dos conectores DB9 para enviar ordenes al dispositivo mediante una conexión RS232.
Un botón de encendido y un fusible, para evitar sobretensiones en el circuito, acompañan al resto de conectores.

Acompañamos en la memoria el manual escaneado para complementar a esta información.

Conexión de los elementos.

Debido a la falta de enchufes disponibles en la azotea del edificio conectamos la fuente de alimentación a una toma de corriente de la tercera planta. Para ello recurrimos a un alargador de 25 metros.

Conectado el controlador a la fuente y la fuente al alargador comenzamos las pruebas del sistema.

Problemas en el dispositivo.

Al encender el dispositivo nos encontramos con el primer problema y es que, al pulsar el interruptor, el sistema no se enciende. En el instante de pulsarlo se escucha un fuerte chasquido que denota que algo falla.

Con un multímetro comprobamos que no tenemos tensión en la toma de corriente. Consultamos con el servicio de mantenimiento del edificio el cual nos informa que el

magnetotérmico que protege el enchufe que estabamos usando ha saltado por una sobrecarga.
Decidimos comprobar el estado del fusible del controlador y éste se encuentra quemado. Nos dirigimos a la tienda mas cercana para comprar un fusible de recambio.

Aunque cambiamos el fusible, el sistema sigue sin responder así que desconectamos el sistema inmediamente para evitar mas daños.

Abriremos el dispositivo para realizar una breve inspección ocular de éste en busca de daños evidentes.

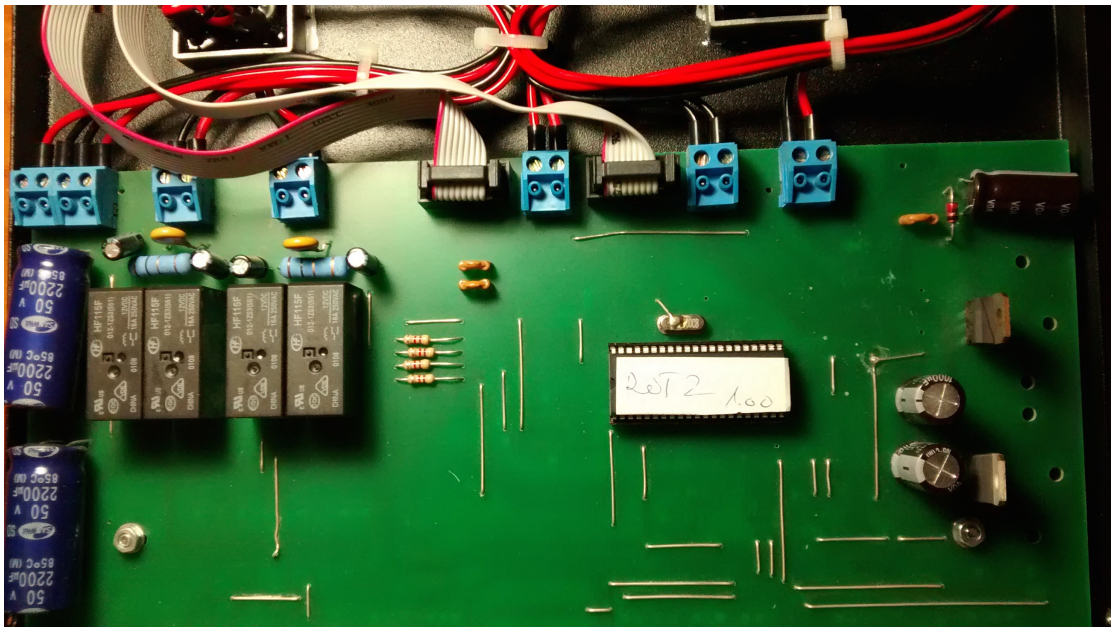


Figura 95. Controlador abierto con la parte superior de la placa al descubierto.

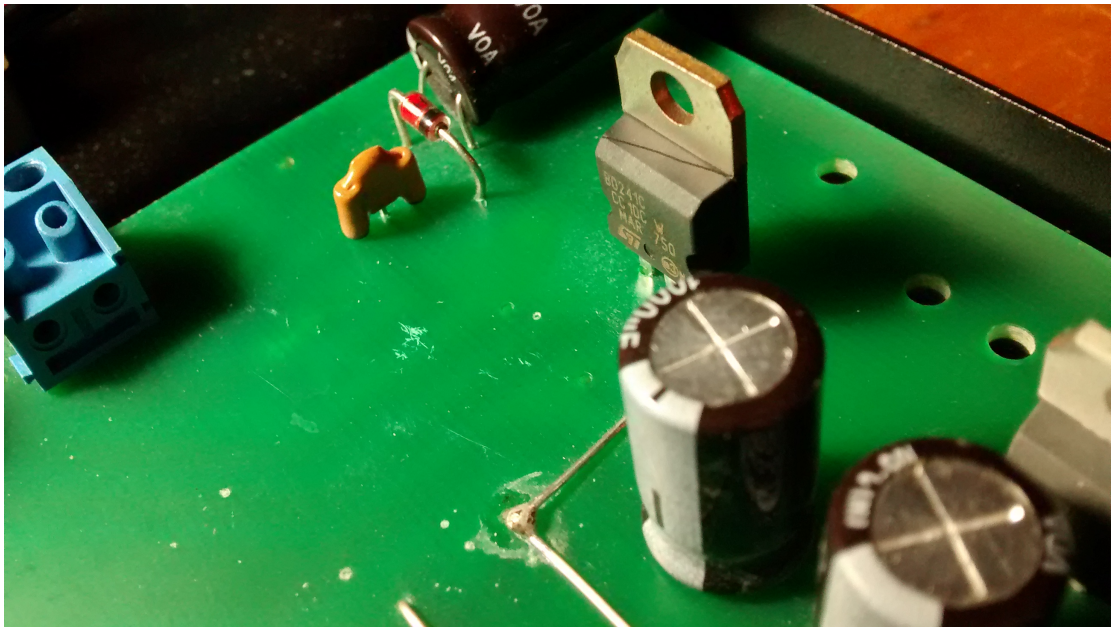


Figura 96. Transistor BJT BD241c.

El único fallo apreciable es el estado de uno de los dos transistores BJT existentes.

Este se encuentra aparentemente quemado.

Sorprende la falta de disipadores en ambos transistores. El estado actual de este se puede deber a este hecho.

Conclusiones.

Aunque los componentes electrónicos del sistema son fácilmente reemplazables se antoja imposible la reparación del mismo sin un estudio de los componentes necesarios.

Simulación del sistema.

Ya que, en un plazo de tiempo razonable, se nos antoja imposible la realización de las pruebas en un entorno real, trataremos de comprobar el funcionamiento del prototipo simulándolo. En este punto explicaremos que software y que hardware será necesario para estas pruebas.

Esquema de conexión.

Utilizaremos el mismo esquema de conexión que en las pruebas destinadas a los convertidores. Nuestra Raspberry Pi ejecutará el programa que hemos creado y enviará las ordenes vía conexión RS232 a nuestro MacBook.

En él tendremos un programa, miniterm, que se encargará de mostrarnos las ordenes. Así podremos comprobar si son correctas o no.

Para simular las condiciones reales el software será ejecutado en remoto.
A través de una conexión de red local nos conectaremos a nuestra Raspberry Pi para ejecutar desde ella el programa. En un entorno real nunca se trabaja junto al sistema de escucha así que, trabajar en remoto, es una de los requisitos imprescindibles de nuestro desarrollo.
Para realizar las conexiones utilizaremos un tunel SSH.

Conexión al sistema remoto.

Una vez conectados los dispositivos siguiendo el esquema del anterior capítulo encendemos nuestra Raspberry Pi.
Ésta demora aproximadamente medio minuto en cargar todo el sistema operativo. Una vez transcurrido ese tiempo nos conectaremos a ella.

Para ello tendremos que ejecutar el siguiente comando.

```
1  ssh -X pi@192.168.2.3
```

Donde -X le indica al programa que deseamos iniciar el entorno gráfico, pi es el usuario que deseamos usar y 192.168.2.3 es la dirección de nuestro dispositivo en nuestra red local.

El sistema nos pedirá la contraseña del usuario. Una vez introducida tendremos total acceso al sistema.

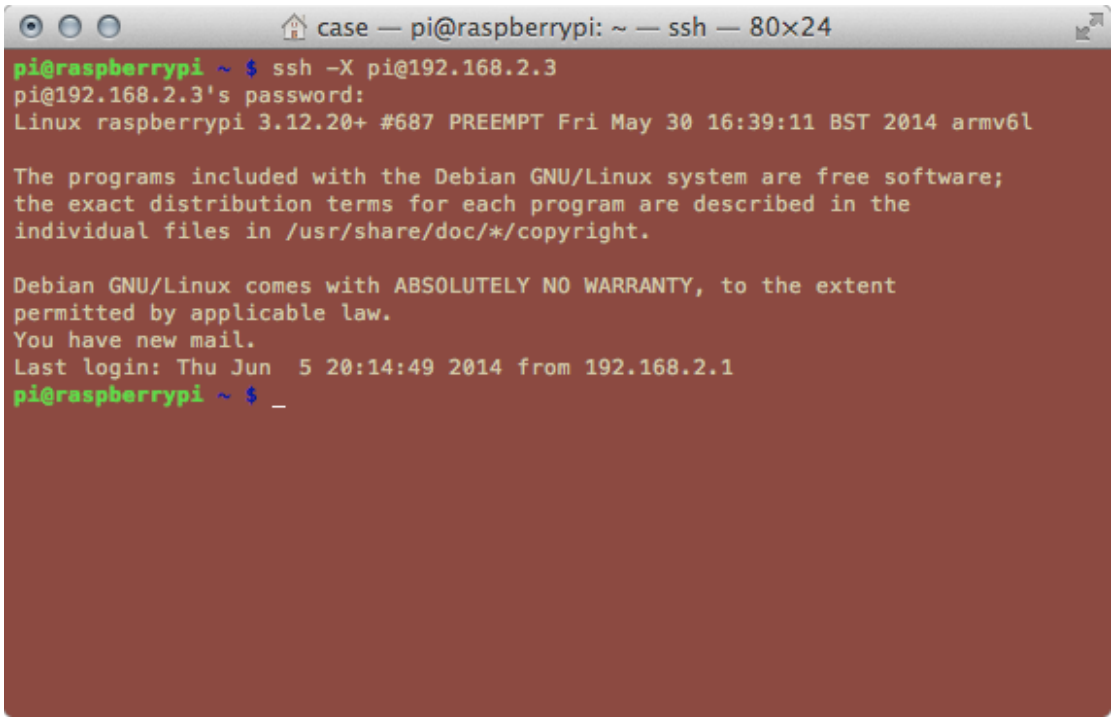


Figura 97. Conexión inicial a la Raspberry Pi.

Desde esta terminal podremos cargar nuestro software. Para ello nos desplazaremos a la carpeta donde se encuentre y ejecutaremos el siguiente comando.

```
1  python main.py
```

Este comando nos dará la siguiente ventana por pantalla.

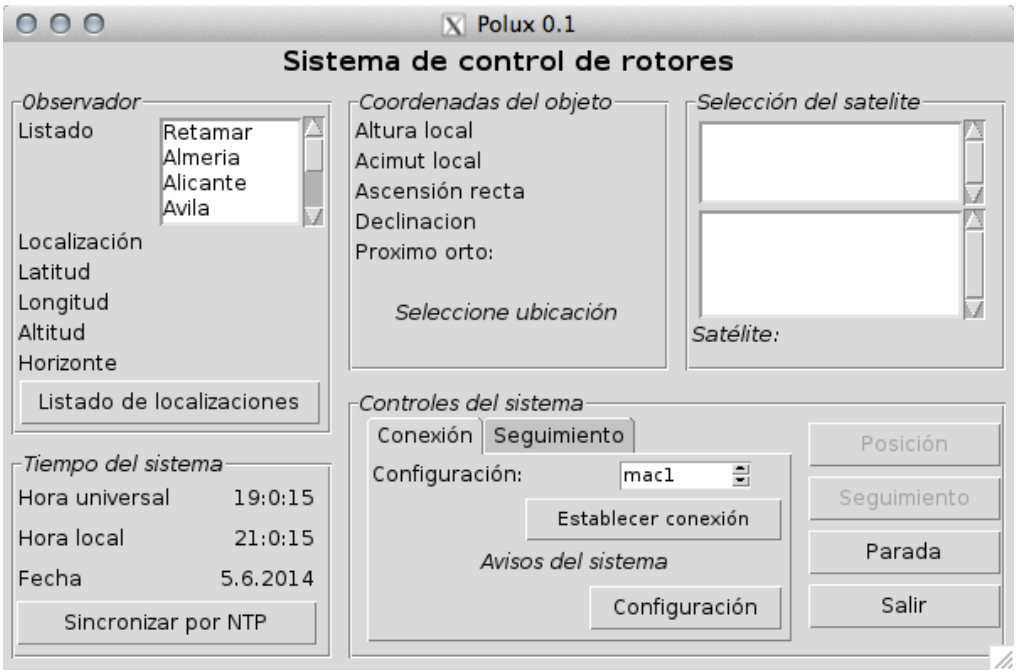


Figura 98. Ventana principal del script ejecutada en remoto.

Como podemos comprobar la conexión es correcta.

Guardar configuración

El primer paso después de iniciar el programa será guardar la configuración de conexión correcta para nuestro caso.

Para ello tendremos que pulsar en el botón "Configuración" y después en la pestaña "Configuración". Con estas acciones obtendremos la siguiente ventana.

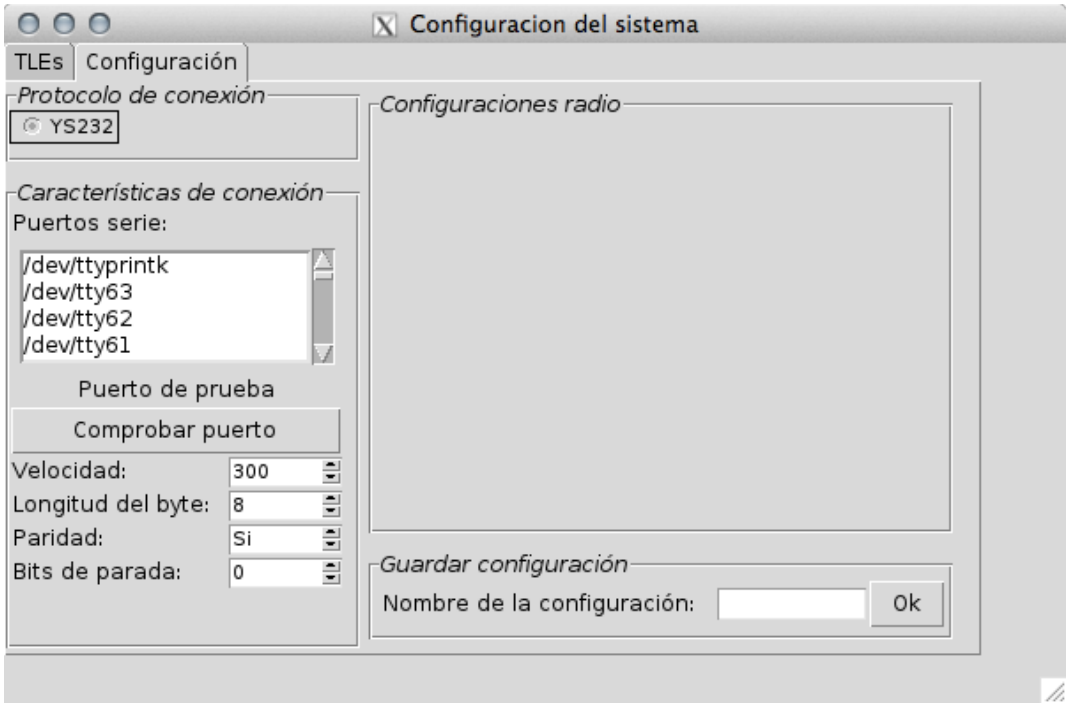


Figura 99. Pestaña Configuración.

En ella seleccionaremos la dirección del puerto, la velocidad de conexión, la longitud del byte, la existencia o no de paridad y los bits de parada. Tendremos también que elegir un nombre que identifique nuestra conexión.

Para esta prueba los valores serán los siguientes:

- Puerto: /dev/ttyAMA0
- Velocidad: 9600 baudios.
- Longitud del byte: 8 bits
- Paridad: No.
- Bits de parada: 1.
- Nombre de la configuración: RasPi

Una vez introduzcamos, o seleccionemos, estos datos pulsaremos el botón Ok.

Uno de los puntos a mejorar de este script es que la nueva configuración no se cargará automaticamente. Tendremos que cerrar el programa y volverlo a abrir para que esta aparezca en el listado de configuraciones posibles.

Una vez abierto comprobaremos si la nueva configuración creada está disponible en el listado. Si está disponible tendremos que seleccionarla con las flechas verticales.

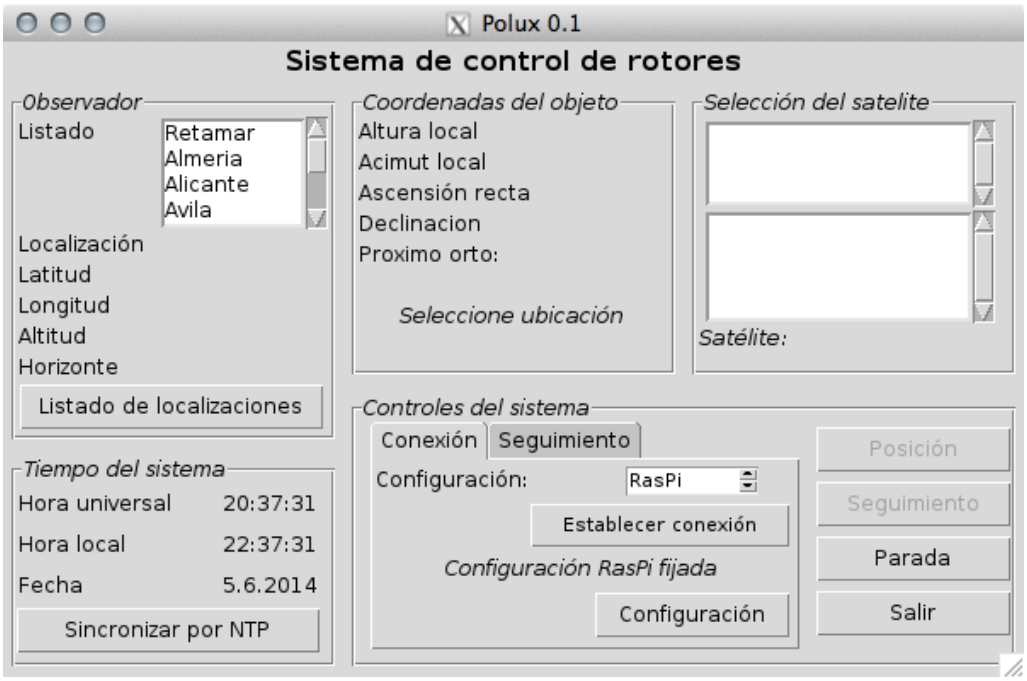


Figura 100. Ventana principal con la configuración RasPi cargada.

La configuración se encuentra cargada y lista para ser usada.

Seguimiento.

Primero seleccionaremos nuestra localización, que será Almería.

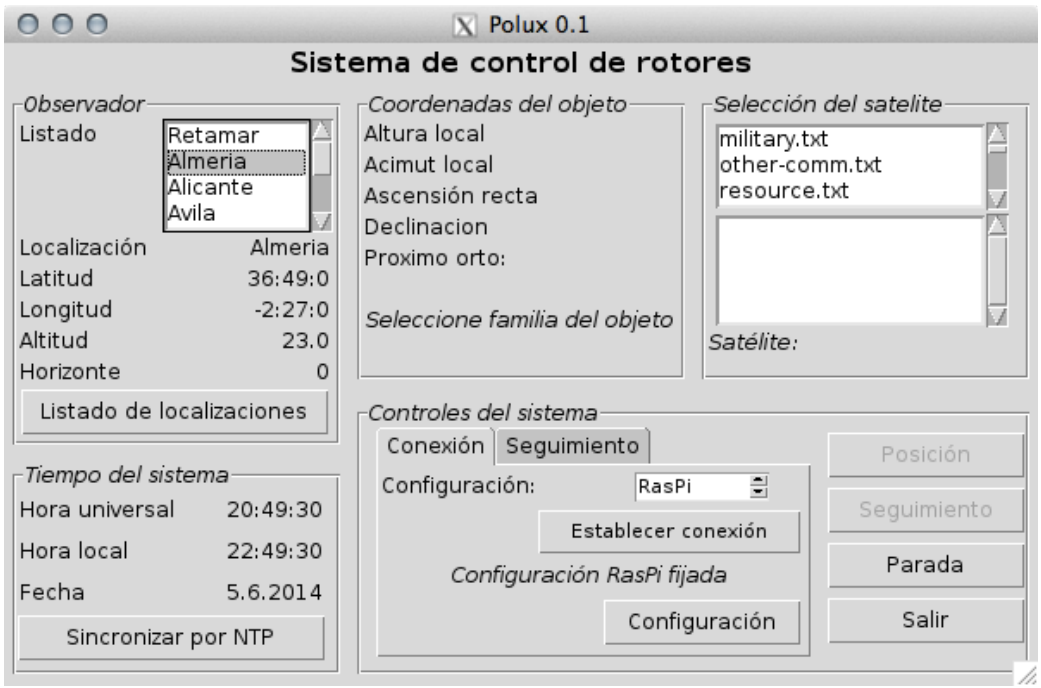


Figura 101. Selección de una localización.

A continuación elegiremos un objeto para activar su seguimiento.

De la familia Molniya escogeremos el satélite Molniya 2-10.

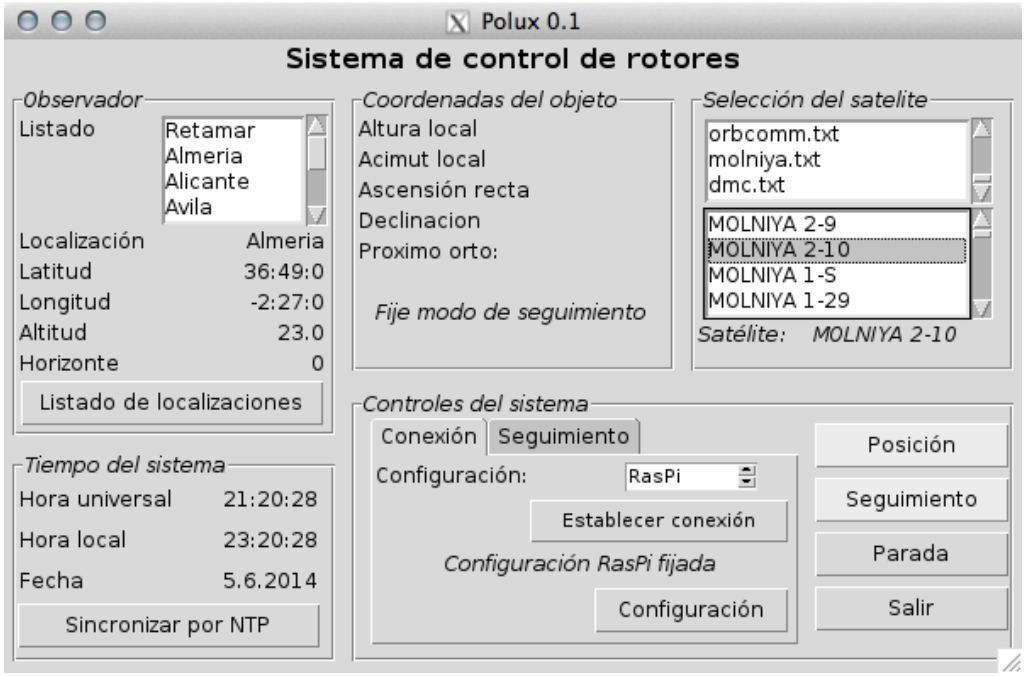


Figura 102. Selección de un satélite.

En este punto tendremos que activar la recepción de cadenas de texto en nuestro Macbook. Para ello ejecutaremos el mismo comando que utilizamos en el capítulo anterior:

```
1  python -m serial.tools.miniterm
```

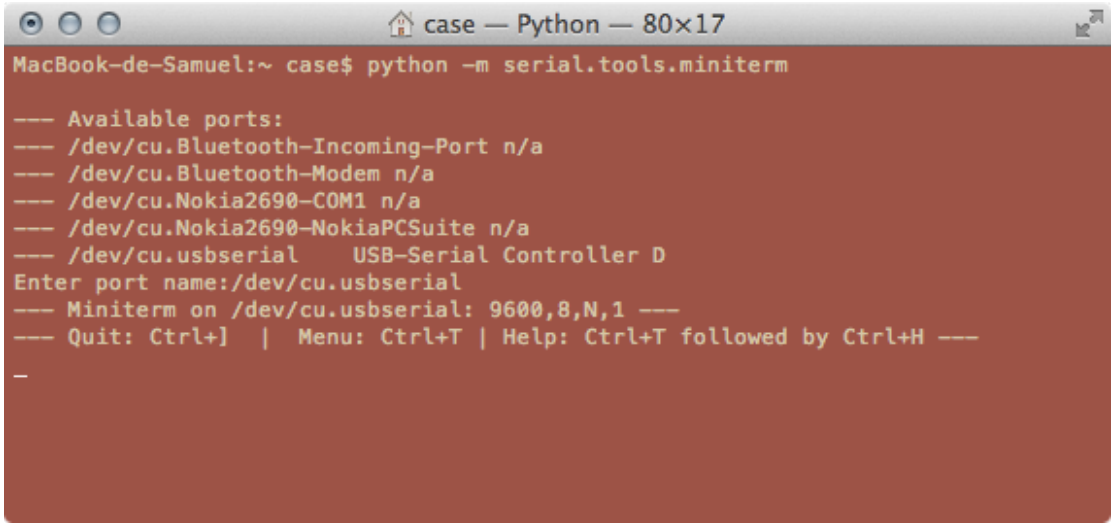


Figura 103. Ventana de recepción de cadenas de texto.

Seleccionando la conexión */dev/cu.usbserial* tendremos listo nuestro sistema para la recepción de mensajes.

El sistema no podrá funcionar hasta que no seleccionemos el modo de seguimiento.
En este punto tenemos dos opciones:

- Modo manual. El cual nos permitirá elegir una velocidad de actualización para las coordenadas que mostraremos por pantalla y otra para las que usaremos con los rotores.
- Modo automático. El cual fijará una velocidad de actualización de un segundo para las coordenadas que se muestran por pantalla y de medio segundo para las destinadas a los rotores.

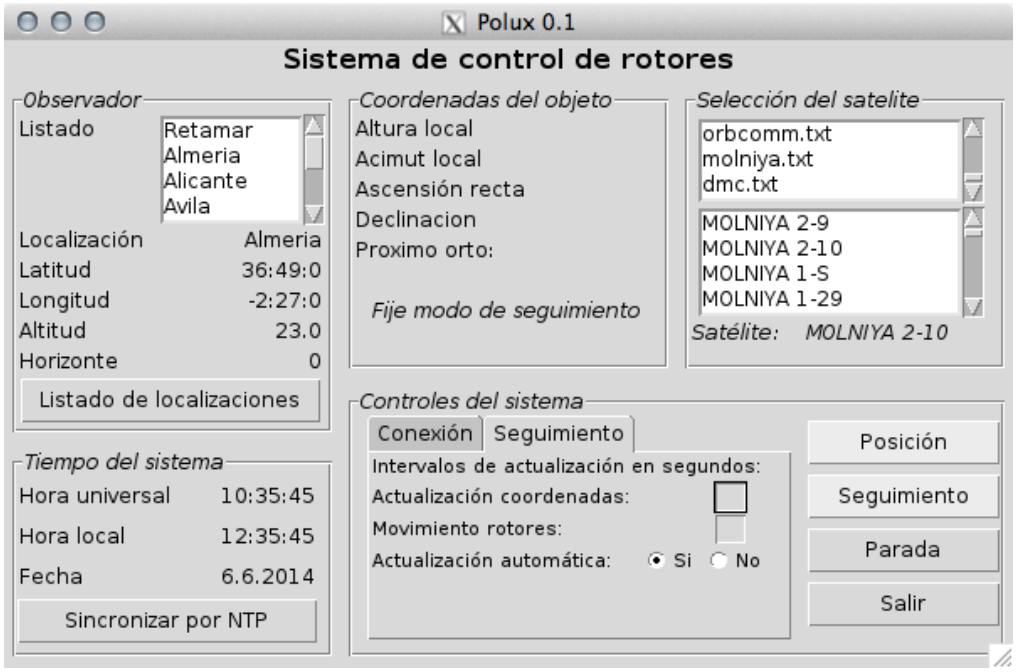


Figura 104. Pestaña de selección del modo de seguimiento.

Como podemos ver en la siguiente figura el seguimiento no se ha iniciado.

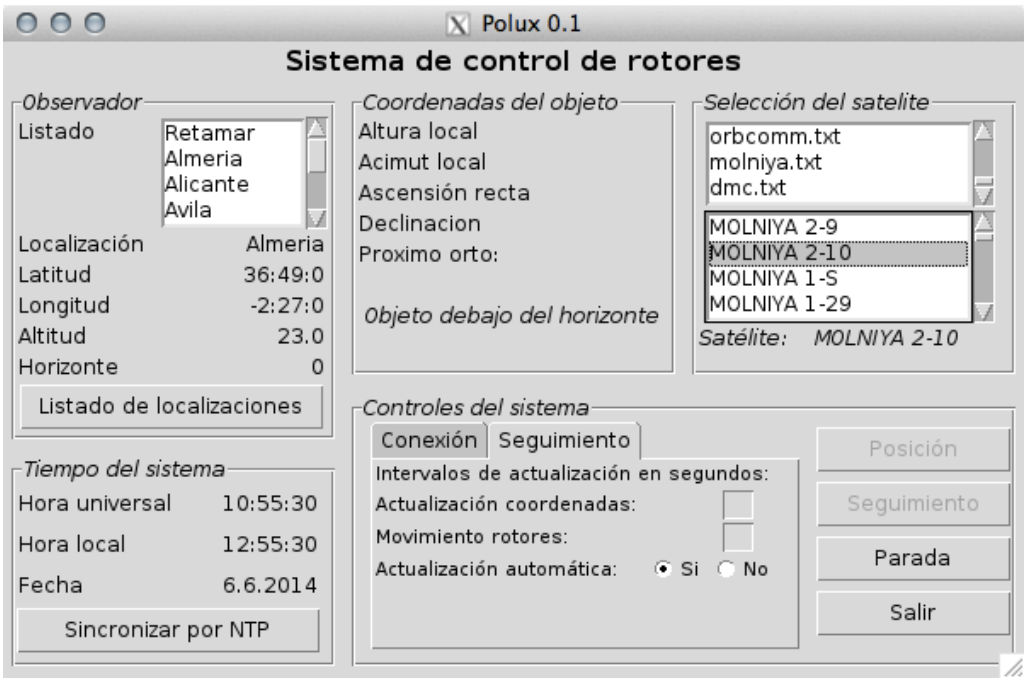


Figura 105. Seguimiento no iniciado por encontrarse el objeto debajo del horizonte.

Acorde a la web Efemérides Astronómicas (Fernandez)[144] las coordenadas del satélite en ese instante son las siguientes:

```
Posición observada:
Azimut(0º=NORTE): 290°38'59"2
Altura : -38°42'36"3 (No visible)
```

Lo cual confirma que nuestro script está funcionando bien.
Probamos con el anterior objeto de la lista, el satélite Molniya 2-9.

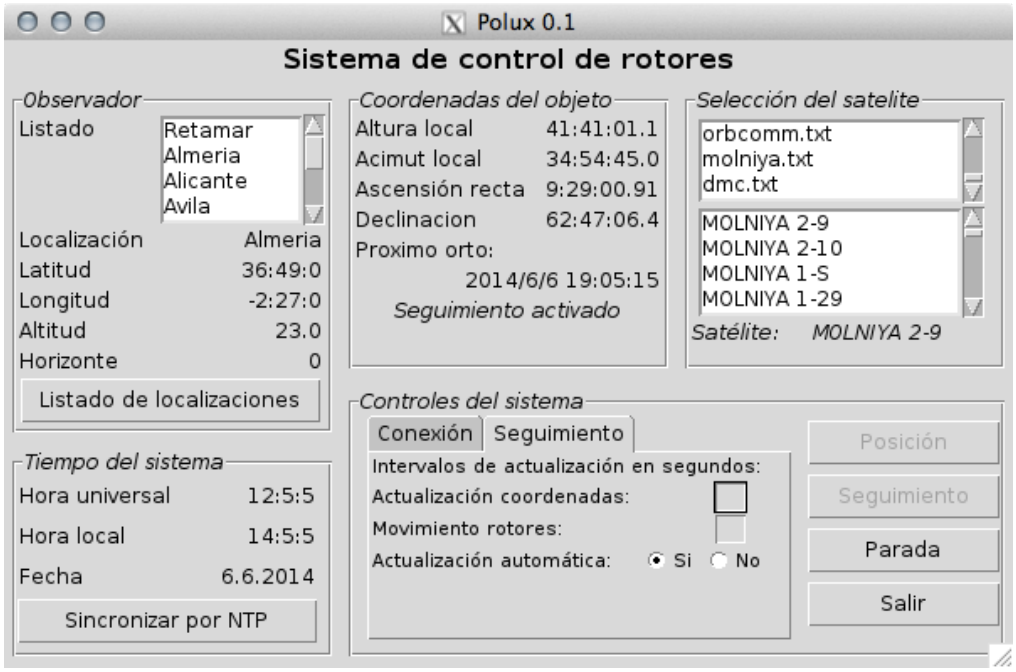


Figura 106. Seguimiento activado.

La orden recibida será la mostrada en la siguiente captura de pantalla.

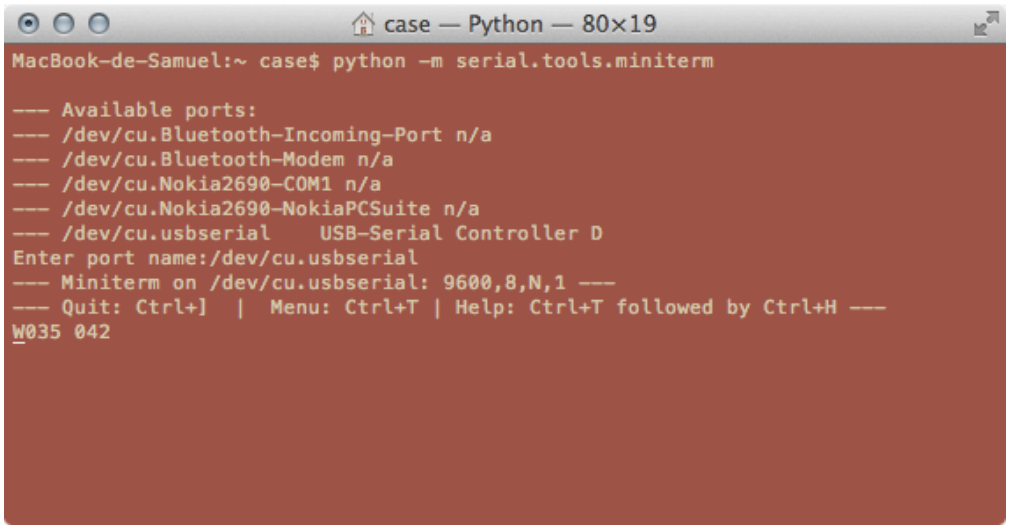


Figura 107. Orden recibida en nuestro MacBook.

[144] http://efemeridesastronomicas.dyndns.org/obj_satelite.htm?norad=7376

Como podemos comprobar el programa ha redondeado los datos antes de enviarlos por el puerto serie. Todo ha funcionado correctamente.
Así ha realizado los siguientes cambios

- Los 34 grados, 54 minutos y 45 segundos de acimut han sido transformados en 35 grados.
- Los 41 grados, 41 minutos y 1,1 segundos de altura han sido transformados en 42 grados.

Parada.

Acorde al protocolo del controlador la parada se implementará enviando el carácter S.

Detendremos el sistema para ver como reacciona.

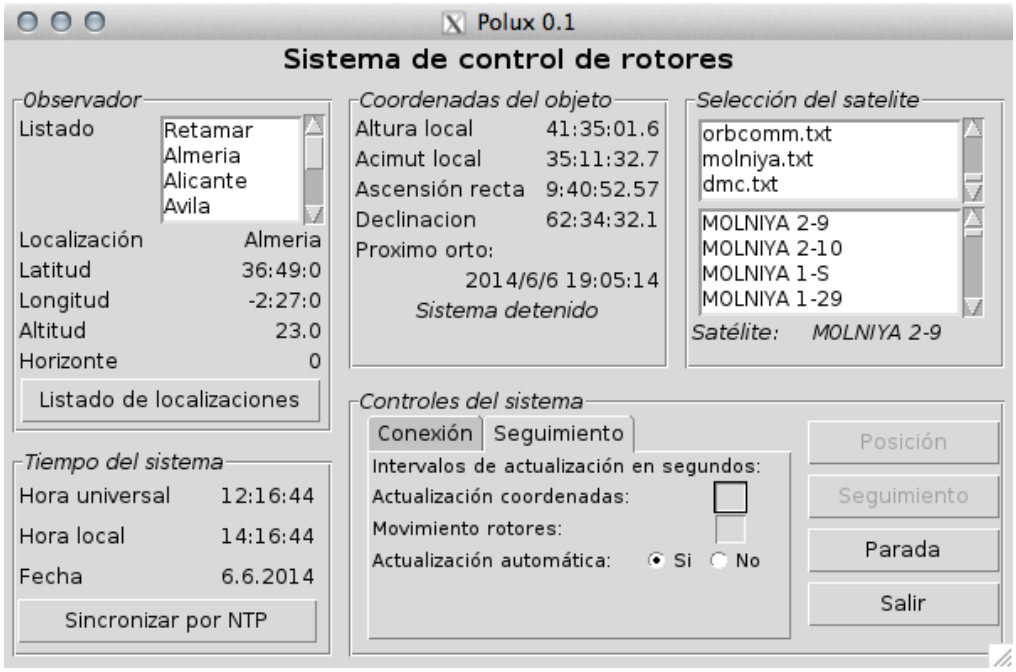


Figura 108. Sistema detenido.

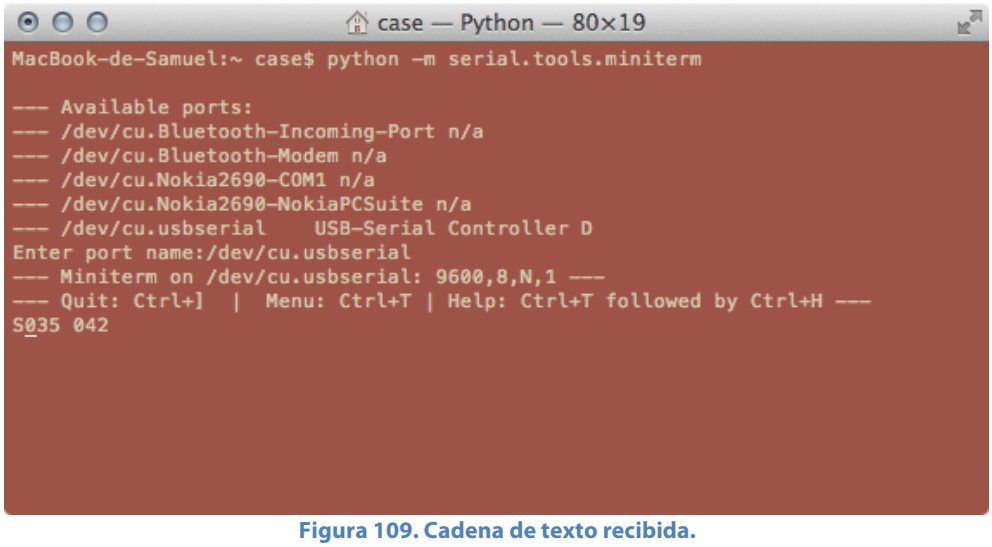


Figura 109. Cadena de texto recibida.

En la figura 1.29 podemos ver como el sistema, una vez pulsado el botón de Parada, muestra el aviso “Sistema detenido”. A la vez que muestra este aviso el sistema envia el carácter “S”.

Este hecho los corroboramos en la figura 8.20 donde vemos como el nuevo carácter recibido es “S”.

Damos por hecho que el funcionamiento del sistema es correcto.

Pasamos a continuación al capítulo nueve. En ese capítulo, el último de esta memoria, enunciaremos algunas de las mejoras posibles del sistema.

9 – Mejoras.

Introducción.

En este capítulo trataremos de dar unas breves pinceladas sobre las mejoras que podríamos aplicar a este prototipo.

Py2exe.

Py2exe (Retziaff, 2014)[145] consiste en un módulo encargado de transformar un script de Python en un ejecutable para Windows.
La ventaja de esto reside en que el usuario de nuestro sistema no tendría porque tener instalado Python en su sistema para poder usar nuestro desarrollo.

Acompañamos en el CD de la memoria una copia de la primera versión de este programa en formato .exe.
Este programa solo ha sido probado en Windows 7 y Windows XP así que puede dar errores inesperados.

Py2app.

Con el mismo planteamiento que el punto anterior nace Py2app. Siguiendo una mecánica de funcionamiento similar podremos crear aplicaciones encapsuladas para los sistemas operativos de la rama Mac OS X.

En el sitio (Ronald Oussoren)[146] web del desarrollo podemos encontrar información sobre este.

También incluimos una primera versión de prueba de una aplicación para Mac 10.9.3 creada a partir de nuestro programa.

Optimización del programa.

Señalaremos algunos puntos a mejorar. Estos influyen en el rendimiento y en la facilidad de uso del programa.

Mas objetos.

Para adaptarse al paradigma de la programación orientada a objetos mas funciones integradas dentro de la clase principal Ventana_principal tendrían que ser transformadas

[145] http://www.py2exe.org/
[146] http://pythonhosted.org/py2app/

en objetos.

Cambios “en caliente”.

Los cambios en las bases de datos de localizaciones de objetos y elementos orbitales solo se producen cuando se cierra y se vuelve abrir el programa.

El código del programa debería modificarse para evitar ese problema.

Mejoras en las visualizaciones.

Este punto está referido a los cambios que podemos implementar para mejorar la usabilidad del programa. En lineas generales son los siguientes:

- Creación de una interfaz donde, sobre un mapa terrestre, se podría ver el paso de los satélites. sobre la superficie del planeta.
- Creación de interfaces gráficas para ver la posición actual de la antena. Actualmente este dato se da por hecho que el mismo que el otorgado a la posición del satélite.

Anexo I – Código fuente.

Rutina principal

Código fuente perteneciente a la rutina principal del programa, con nombre de archivo main.py.

```
# Este archivo usa el encoding: utf-8

################################################################################ ##
# Este archivo main.py es el fichero principal del script.
# En el podemos encontrar la clase Ventana_principal() y las rutinas
# necesarias para el funcionamiento de esta.                                    #
#
# Las líneas sencillas de comentarios separan funciones o marcos.
# Las dobles líneas de comentarios separan las diferentes ventanas.             #
#
# Autor Samuel Góngora García e-mail: s.gongoragarcia@gmail.com                 #
################################################################################

# Módulo necesario para la creación de la interfaz.
import Tkinter
# Módulo necesario para la creación de las tipografías.
import tkFont
# Módulo personalizado para la creación de una lista con desplazamiento.
import scrolledlist
# Módulo para la creación de bases de datos del tipo "Comma Separated Value".
import csv
# Módulos para la gestión de funciones del sistema operativo.
import os
import glob
import sys
# Módulo para la gestión de temas basados en Tkinter.
import ttk
# Módulos para la gestión de hilos de procesamiento.
import threading
import logging
# Módulo necesario para acceder al tiempo del sistema.
import time


# Creación de la clase Ventana_principal().
class Ventana_principal():

    # Función constructora de la clase Ventana_principal.
    # Desde esta función se invocan las rutinas para la  creación de la ventana.
    def __init__(self):

        # Función para crear las tipografías del sistema.
```

```
        self.crear_tipografia()
        # Función destinada a crear los widgets de la ventana.
        self.crear_widgets()
        # Esta función muestra las ciudades disponibles en el widget.
        self.ciudades()
        # Llamo a la función valores_por_defecto() para fijar los valores
        # por defecto del sistema.
        self.valores_por_defecto()

        # A continuación crearemos un hilo para mostrar la hora.
        # Este hilo modificará los atributos de la clase que muestra la hora.
        hora = threading.Thread(target = self.mostrar_hora,
                                name='Hora del sistema')
        # Lo ejecuto como un demonio.
        hora.setDaemon(True)
        # Inicio el hilo en este instante.
        hora.start()

        # Fijamos el primer consejo del sistema.
        self.text_var_eventobueno.set('Seleccione ubicación')

    # Creamos las tipografias del sistema.
    def crear_tipografia(self):

        # Tipografia para Mac OS.
        if sys.platform == 'darwin':
            # Tipografías para las diferentes secciones del programa.
            self.Verd15Bold = tkFont.Font(family = 'Verdana',
                                          size = '15', weight = 'bold')
            self.Verd12Italic = tkFont.Font(family = 'Verdana',
                                            size = '12', slant = 'italic')
            self.Verd13Italic = tkFont.Font(family = 'Verdana',
                                            size = '13', slant = 'italic')
            self.Verd12Roman = tkFont.Font(family = 'Lucida',
                                           size = '12', slant = 'roman')
            self.Verd11Roman = tkFont.Font(family = 'Verdana',
                                           size = '11', slant = 'roman')
        # Tipografia para sistemas basados en Linux.
        elif sys.platform.startswith('linux'):
            # Tipografías para las diferentes secciones del programa.
            self.Verd15Bold = tkFont.Font(family = 'Verdana',
                                          size = '12', weight = 'bold')
            self.Verd12Italic = tkFont.Font(family = 'Verdana',
                                            size = '9', slant = 'italic')
            self.Verd13Italic = tkFont.Font(family = 'Verdana',
                                            size = '10', slant = 'italic')
            self.Verd12Roman = tkFont.Font(family = 'Lucida',
                                           size = '9', slant = 'roman')
            self.Verd11Roman = tkFont.Font(family = 'Verdana',
                                           size = '8', slant = 'roman')
        # Tipografia para Windows.
        elif sys.platform == "win32":
            # Tipografía para las diferentes secciones del programa.
```

```
        self.Verd15Bold = tkFont.Font(family = 'Verdana',
                                      size = '13', weight = 'bold')
        self.Verd12Italic = tkFont.Font(family = 'Verdana',
                                        size = '10', slant = 'italic')
        self.Verd13Italic = tkFont.Font(family = 'Verdana',
                                        size = '11', slant = 'italic')
        self.Verd12Roman = tkFont.Font(family = 'Lucida',
                                       size = '10', slant = 'roman')
        self.Verd11Roman = tkFont.Font(family = 'Verdana',
                                       size = '9', slant = 'roman')
        self.Verd10Roman = tkFont.Font(family = 'Verdana',
                                       size = '8', slant = 'roman')

# Esta función se encarga de crear los widgets para la ventana principal
# del programa.
def crear_widgets(self):

    # Creamos la etiqueta para el encabezado de la interfaz y la asignamos
    # al objeto principal root.
    etiqueta_encabezado = Tkinter.Label(root,\
    text = 'Sistema de control de rotores', font = self.Verd15Bold)
    # Fijamos la etiqueta en el ventana principal del programa.
    etiqueta_encabezado.grid(row = 0, column = 0, rowspan = 1,
                             columnspan = 3)
    # Evitamos que la etiqueta cambie de tamaño.
    etiqueta_encabezado.grid_propagate(0)

    ###########################################################################

    # Creamos el marco 'ventana_observador' para mostrar en el los valores
    # de las diferentes localizaciones.
    # Asignamos este marco a la ventana principal root.
    ventana_observador = Tkinter.LabelFrame(root, text = 'Observador',\
    font = self.Verd12Italic, width=190, height=205)
    # Fijamos el LabelFrame dentro de la ventana del programa.
    ventana_observador.grid(row = 1, column = 0, rowspan = 2,\
    columnspan = 1, padx = 5, pady = 5, sticky = Tkinter.N)
    # Evitamos que el marco cambie de tamaño.
    ventana_observador.grid_propagate(0)

    # Debido a los diferentes tamaños de las tipografías tendremos que
    # fijar diferentes tamaños de columnas y filas.
    # Tamaño de las columnas para Mac OS X.
    if sys.platform == "darwin":
        ventana_observador.columnconfigure(0, minsize = 85)
        ventana_observador.columnconfigure(1, minsize = 85)

    # Tamaño de las columnas para sistemas operativos basados en Linux.
    elif sys.platform.startswith('linux'):
        ventana_observador.columnconfigure(0, minsize = 85)
        ventana_observador.columnconfigure(1, minsize = 85)

    # Tamaño de columnas y filas para sistemas operativos de la familia
```

```
    # Windows.
    elif sys.platform == "win32":
        ventana_observador.columnconfigure(0, minsize = 85)
        ventana_observador.columnconfigure(1, minsize = 85)
        ventana_observador.rowconfigure(1, minsize = 22)
        ventana_observador.rowconfigure(2, minsize = 22)
        ventana_observador.rowconfigure(3, minsize = 22)
        ventana_observador.rowconfigure(4, minsize = 22)
        ventana_observador.rowconfigure(5, minsize = 35)

    # Creamos la etiqueta con el texto 'Listado'.
    etiqueta_busqueda = Tkinter.Label(ventana_observador, text =\
    'Listado', font = self.Verd12Roman)
    # Colocamos la etiqueta en la ventana principal.
    etiqueta_busqueda.grid(row = 0, column = 0, rowspan = 1,\
    columnspan = 1, sticky = Tkinter.NW)
    # Evitamos que la etiqueta cambie de tamaño.
    etiqueta_busqueda.grid_propagate(0)

    # Sistemas operativos Mac OS X.
    if sys.platform == "darwin":
        # Creamos la lista de ciudades en el marco ventana_observador
        # y la vinculamos a la función seleccionar_ciudad.
        self.lista_ciudades = scrolledlist.ScrolledList\
        (ventana_observador, width=10, height=3,\
         callback=self.seleccionar_ciudad)
        # Fijamos la posición de la lista en el marco.
        self.lista_ciudades.grid (row = 0, column = 1, rowspan = 1,
                                   columnspan = 1, sticky = Tkinter.E)

    # Sistemas operativos basados en Linux.
    elif sys.platform.startswith('linux'):
        # Creamos la lista de ciudades en el marco ventana_observador
        # y la vinculamos a la función seleccionar_ciudad.
        self.lista_ciudades = scrolledlist.ScrolledList\
        (ventana_observador, width=10, height=4,\
         callback=self.seleccionar_ciudad)
        # Fijamos la posición de la lista en el marco.
        self.lista_ciudades.grid (row = 0, column = 1, rowspan = 1,
                                   columnspan = 1, sticky = Tkinter.E)

    # Sistemas operativos de la familia Windows.
    elif sys.platform == "win32":
        # Creamos la lista de ciudades en el marco ventana_observador
        # y la vinculamos a la función seleccionar_ciudad.
        self.lista_ciudades = scrolledlist.ScrolledList\
        (ventana_observador, width=11, height=3,\
         callback=self.seleccionar_ciudad)
        # Configuramos la tipografía de la lista de ciudades.
        self.lista_ciudades.listbox.configure(font = self.Verd12Roman)
        # Fijamos la posición de la lista en el marco.
        self.lista_ciudades.grid (row = 0, column = 1, rowspan = 1,
                                   columnspan = 1, sticky = Tkinter.E)
```

```
# Creamos la etiqueta con el texto 'Localización'.
etiqueta_localizacion = Tkinter.Label(ventana_observador,\
text = 'Localización', font = self.Verd12Roman)
# Fijamos la posición de la etiqueta en el marco 'ventana_observador'.
etiqueta_localizacion.grid(row = 1, column = 0, rowspan = 1,
                           columnspan = 1, sticky = Tkinter.W)
# Evitamos que la etiqueta etiqueta_localizacion cambie de tamaño.
etiqueta_localizacion.grid_propagate(0)

# Creamos la StringVar para mostrar la localizacion.
self.text_var_localizacionbuena = Tkinter.StringVar()
# Le asignamos un texto vacio a la StringVar creada anteriormente.
self.text_var_localizacionbuena.set('')

# Creamos la etiqueta para mostrar la localizacion utilizando la
# StringVar anterior.
mostrar_localizacion = Tkinter.Label(ventana_observador,\
textvariable = self.text_var_localizacionbuena, font =\
self.Verd12Roman)
# Le asignamos una posición a la etiqueta mostrar_localizacion en el
# marco 'ventana_observador'.
mostrar_localizacion.grid(row = 1, column = 1, rowspan = 1,
                           columnspan = 1, sticky = Tkinter.E)
# Impedimos que la etiqueta se propague.
mostrar_localizacion.grid_propagate(0)

# Creamos la etiqueta con el texto 'Latitud'.
etiqueta_latitud = Tkinter.Label(ventana_observador, text =\
'Latitud', font = self.Verd12Roman)
# Asignamos una posición a la etiqueta 'Latitud' en el marco.
etiqueta_latitud.grid(row = 2, column = 0, rowspan = 1, columnspan
=1, sticky = Tkinter.W)
# Impedimos que la etiqueta se propague.
etiqueta_latitud.grid_propagate(0)

# Creamos la StringVar para mostrar la latitud.
self.text_var_latitudbuena = Tkinter.StringVar()
# Asignamos un valor vacío a la StringVar anterior.
self.text_var_latitudbuena.set('')

# Creamos una etiqueta para mostrar la latitud mediante la StringVar
# creada anteriormente.
mostrar_latitud = Tkinter.Label(ventana_observador,\
textvariable = self.text_var_latitudbuena, font = self.Verd12Roman)
# Fijamos la posición de la etiqueta en el marco anterior.
mostrar_latitud.grid(row = 2, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.E)
# Evitamos la propagación de la etiqueta.
mostrar_latitud.grid_propagate(0)

# Creamos la etiqueta etiqueta_longitud para mostrar el texto
# 'longitud' en el marco ventana_observador.
```

```
etiqueta_longitud = Tkinter.Label(ventana_observador,\
text = 'Longitud', font = self.Verd12Roman)
# Asignamos la posición de la etiqueta en la ventana anterior.
etiqueta_longitud.grid(row = 3, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)
# Cancelamos el cambio de tamaño de la etiqueta.
etiqueta_longitud.grid_propagate(0)

# Creacion de la StringVar para mostrar la longitud.
self.text_var_longitudbuena = Tkinter.StringVar()
# Ponemos en blanco la StringVar.
self.text_var_longitudbuena.set('')

# Creamos la etiqueta para mostrar la longitud usando el objeto de
# tipo StringVar anterior.
mostrar_longitud = Tkinter.Label(ventana_observador,\
textvariable = self.text_var_longitudbuena, font = self.Verd12Roman)
# Designamos la posición de la etiqueta en el marco.
mostrar_longitud.grid(row = 3, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.E)
# Evitamos el cambio de tamaño de la etiqueta.
mostrar_longitud.grid_propagate(0)

# Creamos la etiqueta etiqueta_altitud con el texto 'Altitud'.
etiqueta_altitud = Tkinter.Label(ventana_observador, text = 'Altitud',
                                 font = self.Verd12Roman)
# Fijamos la etiqueta en el marco 'ventana_observador'.
etiqueta_altitud.grid(row = 4, column = 0, rowspan = 1, columnspan = 1,
                      sticky = Tkinter.W)
# Evitamos la propagación de la etiqueta en el marco.
etiqueta_altitud.grid_propagate(0)

# Creamos una StringVar para mostrar la altitud.
self.text_var_altitudbuena = Tkinter.StringVar()
# Ponemos en blanco el valor de la altitud.
self.text_var_altitudbuena.set('')

# Creamos una etiqueta para mostrar la altitud usando la StringVar
# creada anteriormente.
mostrar_altitud = Tkinter.Label(ventana_observador,\
textvariable = self.text_var_altitudbuena, font = self.Verd12Roman)
# Colocamos la etiqueta creada en el marco 'ventana_observador'.
mostrar_altitud.grid(row = 4, column = 1, rowspan = 1, columnspan = 1,
                     sticky = Tkinter.E)
# Evitamos la propagación de la etiqueta creada.
mostrar_altitud.grid_propagate(0)

# Debido a diferencias de tamaño por los diferentes diseños de la
# interfaz gráfica en cada sistema operativo la visualización del
# horizonte se dará en una posición distinta.
# En sistemas operativos de la familia Windows este dato no será
# proporcionado por pantalla aunque si será accesible mediante línea
# de comandos.
```

```
# Sistemas operativos de la familia Mac OS X.
if sys.platform == 'darwin':
    # Creamos la etiqueta 'Horizonte'.
    etiqueta_horizonte = Tkinter.Label(ventana_observador,\
    text = 'Horizonte', font = self.Verd12Roman)
    # Colocamos la etiqueta 'Horizonte' en el
    # marco 'ventana_observador'.
    etiqueta_horizonte.grid(row = 5, column = 0, rowspan = 1,
                            columnspan = 1, sticky = Tkinter.W)
    # Evitamos el cambio de tamaño de la etiqueta.
    etiqueta_horizonte.grid_propagate(0)

    # Creamos la StringVar para mostrar el valor del horizonte.
    self.text_var_horizontebueno = Tkinter.StringVar()
    # Asignamos un valor nulo a la StringVar anterior.
    self.text_var_horizontebueno.set('')

    # Creamos una etiqueta para mostrar el horizonte utilizando
    # la StringVar creada anteriormente.
    mostrar_horizonte = Tkinter.Label(ventana_observador,\
    textvariable = self.text_var_horizontebueno,\
    font = self.Verd12Roman)
    # Fijamos la etiqueta en el marco ventana_observador.
    mostrar_horizonte.grid(row = 5, column = 1, rowspan = 1,
                           columnspan = 1, sticky = Tkinter.E)
    # Evitamos que la etiqueta se propague.
    mostrar_horizonte.grid_propagate(0)

    # Creamos un entero para asignar una posición a la siguiente
    # etiqueta que necesitemos.
    row_nuevo = 6

# Sistemas operativos basados en Linux.
elif sys.platform.startswith('linux'):
    # Creamos la etiqueta etiqueta_horizonte.
    etiqueta_horizonte = Tkinter.Label(ventana_observador,\
    text = 'Horizonte', font = self.Verd12Roman)
    # Colocamos la etiqueta con el texto 'Horizonte' en el
    # marco ventana_observador.
    etiqueta_horizonte.grid(row = 5, column = 0, rowspan = 1,
                           columnspan = 1, sticky = Tkinter.W)
    # Evitamos el cambio de tamaño de la etiqueta.
    etiqueta_horizonte.grid_propagate(0)

    # Creamos una StringVar para mostrar el horizonte.
    self.text_var_horizontebueno = Tkinter.StringVar()
    # Asignamos un valor nulo a la StringVar creada.
    self.text_var_horizontebueno.set('')

    # Creamos una etiqueta para mostrar el horizonte mediante
    # la StringVar creada anteriormente.
    mostrar_horizonte = Tkinter.Label(ventana_observador,\
    textvariable = self.text_var_horizontebueno,\
```

```
        font = self.Verd12Roman)
        # Colocamos la etiqueta en el marco 'ventana_observador'.
        mostrar_horizonte.grid(row = 5, column = 1, rowspan = 1,
                               columnspan = 1, sticky = Tkinter.E)
        # Evitamos que la etiqueta cambie de tamaño
        mostrar_horizonte.grid_propagate(0)

        # Creamos un entero para asignar una posición a la siguiente
        # etiqueta que necesitemos.
        row_nuevo = 6

    # Como no crearemos ninguna etiqueta para el horizonte directamente
    # fijaremos el valor de la fila siguiente en Windows.
    elif sys.platform == "win32":
        # Valor de la fila.
        row_nuevo = 5

    # Creamos el botón para acceder al listado de localizaciones.
    boton_ciudades = Tkinter.Button(ventana_observador,\
    text = 'Listado de localizaciones', width = 19,\
    font = self.Verd12Roman, command = self.listado_posiciones)
    # Colocamos el botón en el marco ventana_observador.
    # Utilizamos el valor de row_nuevo para la fila de este botón.
    boton_ciudades.grid(row = row_nuevo, column = 0, rowspan = 1,
                        columnspan = 2)
    # Evitamos que el botón cambie de tamaño
    boton_ciudades.grid_propagate(0)

    ###########################################################################

    # Creamos el marco ventana_tiempo para mostrar en el la hora actual
    # del sistema.
    # También colocamos este marco
    ventana_tiempo = Tkinter.LabelFrame(root, text = 'Tiempo del sistema',\
    font = self.Verd12Italic, width=190, height = 123)
    # Evitamos la propagación del marco dentro de la ventana principal.
    ventana_tiempo.grid_propagate(0)
    # Fijamos una posición para el marco ventana_tiempo dentro de
    # la ventana principal.
    ventana_tiempo.grid(row = 3, column = 0, rowspan = 1, columnspan = 1,
                        padx = 5, pady = 5, sticky = Tkinter.N)

    # Asignaremos diferentes tamaños mínimos a las filas y columnas del
    # marco dependiendo del sistema operativo.
    # Configuración para sistemas operativos de la familia Macintosh.
    if sys.platform == "darwin":
        # Asignamos un tamaño mínimo de 24 píxeles a la primera fila.
        ventana_tiempo.rowconfigure(0, minsize = 24)
        # Asignamos un tamaño mínimo de 24 píxeles a la segunda fila.
        ventana_tiempo.rowconfigure(1, minsize = 24)
        # Asignamos un tamaño mínimo de 24 píxeles a la tercera fila.
        ventana_tiempo.rowconfigure(2, minsize = 24)
```

```
# Valores para sistemas operativos basados en Linux.
elif sys.platform.startswith('linux'):
    # Fijamos un tamaño mínimo de 24 píxeles a la primera fila.
    ventana_tiempo.rowconfigure(0, minsize = 24)
    # Fijamos un tamaño mínimo de 24 píxeles a la segunda fila.
    ventana_tiempo.rowconfigure(1, minsize = 24)
    # Fijamos un tamaño mínimo de 24 píxeles a la tercera fila.
    ventana_tiempo.rowconfigure(2, minsize = 24)

# Valores para sistemas operativos de la familia Windows.
elif sys.platform == "win32":
    # Asignamos un tamaño mínimo de 24 píxeles a la primera fila.
    ventana_tiempo.rowconfigure(0, minsize = 24)
    # Asignamos un tamaño mínimo de 24 píxeles a la segunda fila.
    ventana_tiempo.rowconfigure(1, minsize = 24)
    # Asignamos un tamaño
    ventana_tiempo.rowconfigure(2, minsize = 24)
    ventana_tiempo.columnconfigure(0, minsize = 125)
    ventana_tiempo.columnconfigure(1, minsize = 60)

# Creamos una etiqueta con el texto 'Hora universal'.
etiqueta_hora = Tkinter.Label(ventana_tiempo, text = 'Hora universal',
                              font = self.Verd12Roman)
# Fijamos la posición de la etiqueta en el marco ventana_tiempo.
etiqueta_hora.grid(row = 0, column = 0, rowspan = 1, columnspan = 1,
                   sticky = Tkinter.W)
# Evitamos el cambio de tamaño de la etiqueta
etiqueta_hora.grid_propagate(0)

# Creamos la StringVar text_var_horabuena para mostrar la hora actual
# del sistema.
self.text_var_horabuena = Tkinter.StringVar()
# Ponemos en blanco la StringVar creada.
self.text_var_horabuena.set('')

# Creamos la etiqueta mostrar_hora para mostrar el tiempo mediante la
# StringVar anterior.
mostrar_hora = Tkinter.Label(ventana_tiempo,\
textvariable = self.text_var_horabuena, font = self.Verd12Roman)
# Colocamos la etiqueta en el marco.
mostrar_hora.grid(row = 0, column = 1, rowspan = 1, columnspan = 1,
                  sticky = Tkinter.E)
# Evitamos que la etiqueta cambie de forma.
mostrar_hora.grid_propagate(0)

# Creamos una etiqueta llamada etiqueta_horalocal con el texto
# 'Hora local'.
etiqueta_horalocal = Tkinter.Label(ventana_tiempo, text = 'Hora local',
                                   font = self.Verd12Roman)
# Colocamos el etiqueta creada en el marco 'ventana_tiempo'
etiqueta_horalocal.grid(row = 1, column = 0, rowspan = 1,
                        columnspan = 1, sticky = Tkinter.W)
# Cancelamos la propagación de la etiqueta.
```

```
etiqueta_horalocal.grid_propagate(0)

# Creamos la StringVar text_var_horalocalbuena para mostrar los
# valores de la hora local actualizados.
self.text_var_horalocalbuena = Tkinter.StringVar()
# Asignamos a la StringVar un valor nulo
self.text_var_horalocalbuena.set('')

# Creamos la etiqueta mostrar_horalocal para visualizar los valores de
# la hora local mediante la StringVar anterior.
mostrar_horalocal = Tkinter.Label(ventana_tiempo,\
textvariable = self.text_var_horalocalbuena, font = self.Verd12Roman)
# Fijamos la posición de la etiqueta en el marco.
mostrar_horalocal.grid(row = 1, column = 1, rowspan = 1,
                       columnspan = 1, sticky = Tkinter.E)
# Evitamos la propagación de la etiqueta.
mostrar_horalocal.grid_propagate(0)

# Creamos una etiqueta para mostrar el texto 'Fecha del sistema'.
# Debido a las cambios entre distintos sistemas operativos tendremos
# que definir tres etiquetas diferentes.
# Sistemas operativos de la familia Macintosh
if sys.platform == "darwin":
    # Creamos una etiqueta para mostrar el texto 'Fecha del sistema'.
    etiqueta_fecha = Tkinter.Label(ventana_tiempo,\
    text = 'Fecha del sistema', font = self.Verd12Roman)
    # Colocamos la etiqueta en su posición del marco.
    etiqueta_fecha.grid(row = 2, column = 0, rowspan = 1,
                        columnspan = 1, sticky = Tkinter.W)
    # Evitamos la propagación de la etiqueta.
    etiqueta_fecha.grid_propagate(0)

# Sistemas operativos basados en Linux.
elif sys.platform.startswith('linux'):
    # Creamos la etiqueta etiqueta_fecha con el texto 'Fecha'.
    etiqueta_fecha = Tkinter.Label(ventana_tiempo, text = 'Fecha',
                                   font = self.Verd12Roman)
    # Colocamos la etiqueta en el marco.
    etiqueta_fecha.grid(row = 2, column = 0, rowspan = 1,
                        columnspan = 1, sticky = Tkinter.W)
    # Evitamos la propagación de la etiqueta.
    etiqueta_fecha.grid_propagate(0)

# Sistemas operativos de la familia Windows.
elif sys.platform == "win32":
    # Creamos la etiqueta etiqueta_fecha con el texto 'Fecha'.
    etiqueta_fecha = Tkinter.Label(ventana_tiempo, text = 'Fecha',
                                   font = self.Verd12Roman)
    # Fijamos la posición de la etiqueta en el marco.
    etiqueta_fecha.grid(row = 2, column = 0, rowspan = 1,
                        columnspan = 1, sticky = Tkinter.W)
    # Evitamos el cambio de tamaño de la etiqueta.
    etiqueta_fecha.grid_propagate(0)
```

```
# Creamos una Stringvar text_var_fechabuena para mostrar la fecha
# actual del sistema.
self.text_var_fechabuena = Tkinter.StringVar()
# Ponemos la fecha en blanco.
self.text_var_fechabuena.set('')

# Creamos la etiqueta mostrar_fecha para mostrar la fecha mediante la
# StringVar anterior.
mostrar_fecha = Tkinter.Label(ventana_tiempo,\
textvariable = self.text_var_fechabuena, font = self.Verd12Roman)
# Colocamos la etiqueta en el marco.
mostrar_fecha.grid(row = 2, column = 1, rowspan = 1, columnspan = 1,
                   sticky = Tkinter.E)
# Evitamos la propagación de la etiqueta por el marco.
mostrar_fecha.grid_propagate(0)

# Creamos un boton para llamar a la rutina de sincronización de la hora
# mediante NTP.
# Sistemas operativos de la familia Mac OS X.
if sys.platform == "darwin":
    # Creamos un botón con el texto 'Sincronizar mediante NTP'.
    boton_sincronizar = Tkinter.Button(ventana_tiempo,\
    text = 'Sincronizar mediante NTP', font = self.Verd12Roman,\
    command = self.sincronizar_hora)
    # Colocamos el botón en el marco.
    boton_sincronizar.grid(row = 3, column = 0, rowspan = 1,
                           columnspan = 2, sticky = Tkinter.S)
    # Evitamos el cambio de tamaño del botón dentro del marco.
    boton_sincronizar.grid_propagate(0)

# Sistemas operativos basados en Linux.
elif sys.platform.startswith('linux'):
    # Creamos el boton boton_sincronizar con el texto 'Sincronizar por
    # NTP'.
    boton_sincronizar = Tkinter.Button(ventana_tiempo,\
    text = 'Sincronizar por NTP', font = self.Verd12Roman,\
    command = self.sincronizar_hora, width = 19)
    # Colocamos el botón en el marco ventana_tiempo.
    boton_sincronizar.grid(row = 3, column = 0, rowspan = 1,
                           columnspan = 2, sticky = Tkinter.S)
    # Evitamos la propagación del botón en el marco.
    boton_sincronizar.grid_propagate(0)

# Sistemas operativos de la familia Windows.
elif sys.platform == "win32":
    # Creamos el boton boton_sincronizar con el texto 'Sincronizar por
    # NTP'.
    boton_sincronizar = Tkinter.Button(ventana_tiempo,\
    text = 'Sincronizar por NTP', font = self.Verd12Roman,\
    command = self.sincronizar_hora, width = 19)
    # Colocamos el botón en el marco ventana_tiempo.
    boton_sincronizar.grid(row = 3, column = 0, rowspan = 1,
```

```
                                columnspan = 2, sticky = Tkinter.S)
    # Evitamos el cambio de tamaño del botón.
    boton_sincronizar.grid_propagate(0)

########################################################################

# Creamos el marco ventana_localizacion para mostrar en tiempo real
# la posición del satélite elegido.
ventana_localizacion = Tkinter.LabelFrame(root,\
text = 'Coordenadas del objeto', font = self.Verd12Italic,\
width=190, height = 167)
# Evitamos el cambio de tamaño del marco dentro de la ventana.
ventana_localizacion.grid_propagate(0)
# Colocamos el marco ventana_localizacion en la ventana principal.
ventana_localizacion.grid(row = 1, column = 1, rowspan = 2,\
columnspan = 1, padx = 5, pady = 5, sticky = Tkinter.N)

# Fijamos el valor mínimo de columnas y filas para los tres sistemas
# operativos soportados.
# Sistemas operativos de la familia Macintosh.
if sys.platform == "darwin":
    ventana_localizacion.columnconfigure(0, minsize = 105)
    ventana_localizacion.columnconfigure(1, minsize = 80)

# Sistemas operativos basados en Linux.
elif sys.platform.startswith('linux'):
    ventana_localizacion.columnconfigure(0, minsize = 105)
    ventana_localizacion.columnconfigure(1, minsize = 80)

# Sistemas operativos de la familia Windows.
elif sys.platform == "win32":
    ventana_localizacion.rowconfigure(0, minsize = 24)
    ventana_localizacion.rowconfigure(1, minsize = 24)
    ventana_localizacion.rowconfigure(2, minsize = 24)
    ventana_localizacion.rowconfigure(3, minsize = 24)
    ventana_localizacion.rowconfigure(4, minsize = 24)
    ventana_localizacion.rowconfigure(5, minsize = 24)

# A continuación tenemos una sentencia if-elif destinada a seleccionar
# como se crearán los elementos del marco ventana_localizacion.
# Sistemas operativos de la familia Macintosh
if sys.platform == "darwin":
    # Creamos una etiqueta con el texto 'Altura local'.
    etiqueta_altura = Tkinter.Label(ventana_localizacion,\
    text = 'Altura local', font = self.Verd12Roman)
    # Colocamos la etiqueta en el marco ventana_localizacion.
    etiqueta_altura.grid(row = 0, column = 0, rowspan = 1,
                         columnspan = 1, sticky = Tkinter.W)
    # Evitamos la propagación de la etiqueta.
    etiqueta_altura.grid_propagate(0)

    # Creamos una StringVar para mostrar la altura.
    self.text_var_alturabuena = Tkinter.StringVar()
```

```
# Ponemos en blanco la StringVar creada.
self.text_var_alturabuena.set('')

# Creamos la etiqueta mostrar_altura para mostrar la altura
# mediante la StringVar anterior.
mostrar_altura = Tkinter.Label(ventana_localizacion,\
textvariable = self.text_var_alturabuena, font = self.Verd12Roman)
# Colocamos la etiqueta creada en el marco ventana_localizacion.
mostrar_altura.grid(row = 0, column = 1, rowspan = 1,
                    columnspan = 1, sticky = Tkinter.E)
# Evitamos la propagación de la etiqueta.
mostrar_altura.grid_propagate(0)

# Creamos la etiqueta etiqueta_acimut con el texto 'Acimut local'.
etiqueta_acimut = Tkinter.Label(ventana_localizacion,\
text = 'Acimut local', font = self.Verd12Roman)
# Fijamos la posición de la etiqueta etiqueta_acimut.
etiqueta_acimut.grid(row = 1, column = 0, rowspan = 1,
                     columnspan = 1, sticky = Tkinter.W)
# Evitamos el cambio de tamaño de la etiqueta.
etiqueta_acimut.grid_propagate(0)

# Creamos la StringVar text_var_acimutbueno para mostrar el acimut.
self.text_var_acimutbueno = Tkinter.StringVar()
# Ponemos en blanco el objeto creado anteriormente.
self.text_var_acimutbueno.set('')

# Creamos la etiqueta mostrar_acimut para visualizar el acimut
# utilizando la StringVar anterior.
mostrar_acimut = Tkinter.Label(ventana_localizacion,\
textvariable = self.text_var_acimutbueno, font = self.Verd12Roman)
# Asignamos una posición en el marco a la etiqueta anterior.
mostrar_acimut.grid(row = 1, column = 1, rowspan = 1,
                    columnspan = 1, sticky = Tkinter.E)
# Evitamos la propagación de la etiqueta.
mostrar_acimut.grid_propagate(0)

# Creamos una etiqueta con el texto 'Ascensión recta'.
etiqueta_ascension = Tkinter.Label(ventana_localizacion,\
text = 'Ascensión recta', font = self.Verd12Roman)
# Colocamos la etiqueta etiqueta_ascension en el
# marco ventana_localizacion
etiqueta_ascension.grid(row = 2, column = 0, rowspan = 1,
                        columnspan = 1, sticky = Tkinter.W)
# Evitamos el cambio de tamaño de la etiqueta etiqueta_ascension
etiqueta_ascension.grid_propagate(0)

# Creamos una StringVar para mostrar la ascensión recta.
self.text_var_ascensionbuena = Tkinter.StringVar()
# Asignamos un valor nulo a la StringVar creada.
self.text_var_ascensionbuena.set('')

# Creamos una etiqueta para mostrar la ascensión recta usando
```

```
# la StringVar anterior.
mostrar_ascension = Tkinter.Label(ventana_localizacion,\
textvariable = self.text_var_ascensionbuena,\
font = self.Verd12Roman)
# Fijamos la posición de la etiqueta creada.
mostrar_ascension.grid(row = 2, column = 1, rowspan = 1,
                       columnspan = 1, sticky = Tkinter.E)
# Evitamos el cambio de tamaño de la etiqueta anterior.
mostrar_ascension.grid_propagate(0)

# Creamos la etiqueta etiqueta_declinacion y le asignamos
# el texto 'Declinacion'.
etiqueta_declinacion = Tkinter.Label(ventana_localizacion,\
text = 'Declinacion', font = self.Verd12Roman)
# Fijamos la posición de la etiqueta etiqueta_declinacion.
etiqueta_declinacion.grid(row = 3, column = 0, rowspan = 1,
                          columnspan = 1, sticky = Tkinter.W)
# Evitamos la propagación de la etiqueta.
etiqueta_declinacion.grid_propagate(0)

# Creamos el objeto text_var_declinacionbuena
# para mostrar la declinación.
self.text_var_declinacionbuena = Tkinter.StringVar()
# Ponemos en blanco la StringVar.
self.text_var_declinacionbuena.set('')

# Creamos la etiqueta mostrar_declinacion para visualizar la
# declinacion mediante la StringVar text_var_declinacionbuena
mostrar_declinacion = Tkinter.Label(ventana_localizacion,\
textvariable = self.text_var_declinacionbuena,\
font = self.Verd12Roman)
# Asignamos una posición a la etiqueta mostrar_declinacion
mostrar_declinacion.grid(row = 3, column = 1, rowspan = 1,
                         columnspan = 1, sticky = Tkinter.E)
# Evitamos el cambio de tamaño de la etiqueta anterior.
mostrar_declinacion.grid_propagate(0)

# Creamos una etiqueta con el texto 'Próximo orto:'.
etiqueta_orto = Tkinter.Label(ventana_localizacion,\
text = 'Próximo orto:', font = self.Verd12Roman)
# Colocamos la etiqueta en el marco ventana_localizacion.
etiqueta_orto.grid(row = 4, column = 0, rowspan = 1,
                   columnspan = 2, sticky = Tkinter.W)
# Impedimos que esta etiqueta cambie de tamaño.
etiqueta_orto.grid_propagate(0)

# Creamos una StringVar para mostrar el próximo orto del objeto
# elegido.
self.text_var_ortobueno = Tkinter.StringVar()
# Ponemos el texto de la StringVar en blanco.
self.text_var_ortobueno.set('')

# Creamos una etiqueta donde mostrar el próximo orto
```

```
        mostrar_orto = Tkinter.Label(ventana_localizacion,\
        textvariable = self.text_var_ortobueno, font = self.Verd12Roman)
        # Colocamos la etiqueta en el marco ventana_localizacion.
        mostrar_orto.grid(row = 5, column = 0, rowspan = 1,
                          columnspan = 2, sticky = Tkinter.E)
        # Evitamos que la etiqueta cambie de tamaño.
        mostrar_orto.grid_propagate(0)

        # Creamos una StringVar para mostrar los eventos del sistema.
        self.text_var_eventobueno = Tkinter.StringVar()
        # Dejamos el texto de la StringVar recién creada en blanco.
        self.text_var_eventobueno.set('')

        # Creamos una etiqueta para mostrar eventos del sistema utilizando
        # la StringVar anterior.
        mostrar_evento = Tkinter.Label(ventana_localizacion,\
        textvariable = self.text_var_eventobueno, font = self.Verd12Italic)
        # Colocamos la etiqueta en el marco ventana_localizacion.
        mostrar_evento.grid(row = 6, column = 0, rowspan = 1,
                            columnspan = 2)
        # Evitamos que la etiqueta cambie de tamaño.
        mostrar_evento.grid_propagate(0)

    # En el caso de que estemos trabajando con Linux el programa saltará
    # a esta rutina.
    elif sys.platform.startswith('linux'):
        # Creamos una etiqueta con el texto 'Altura local'.
        etiqueta_altura = Tkinter.Label(ventana_localizacion,\
        text = 'Altura local', font = self.Verd12Roman)
        # Colocamos la etiqueta en el marco ventana_localizacion.
        etiqueta_altura.grid(row = 0, column = 0, rowspan = 1,
                             columnspan = 1, sticky = Tkinter.W)
        # Evitamos la propagación de la etiqueta.
        etiqueta_altura.grid_propagate(0)

        # Creamos una StringVar para mostrar la altura.
        self.text_var_alturabuena = Tkinter.StringVar()
        # Ponemos en blanco la StringVar creada.
        self.text_var_alturabuena.set('')

        # Creamos la etiqueta mostrar_altura para mostrar la altura
        # mediante la StringVar anterior.
        mostrar_altura = Tkinter.Label(ventana_localizacion,\
        textvariable = self.text_var_alturabuena, font = self.Verd12Roman)
        # Colocamos la etiqueta creada en el marco ventana_localizacion.
        mostrar_altura.grid(row = 0, column = 1, rowspan = 1,\
        columnspan = 1, sticky = Tkinter.E)
        # Evitamos la propagación de la etiqueta.
        mostrar_altura.grid_propagate(0)

        # Creamos la etiqueta etiqueta_acimut con el texto 'Acimut local'.
        etiqueta_acimut = Tkinter.Label(ventana_localizacion,\
        text = 'Acimut local', font = self.Verd12Roman)
```

```
# Fijamos la posición de la etiqueta etiqueta_acimut.
etiqueta_acimut.grid(row = 1, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)
# Evitamos el cambio de tamaño de la etiqueta
etiqueta_acimut.grid_propagate(0)

# Creamos la StringVar text_var_acimutbueno para mostrar el acimut.
self.text_var_acimutbueno = Tkinter.StringVar()
# Ponemos en blanco el objeto creado anteriormente.
self.text_var_acimutbueno.set('')

# Creamos la etiqueta mostrar_acimut para visualizar el acimut
# utilizando la StringVar anterior.
mostrar_acimut = Tkinter.Label(ventana_localizacion,\
textvariable = self.text_var_acimutbueno, font = self.Verd12Roman)
# Asignamos una posición en el marco a la etiqueta anterior.
mostrar_acimut.grid(row = 1, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.E)
# Evitamos la propagación de la etiqueta.
mostrar_acimut.grid_propagate(0)

# Creamos una etiqueta con el texto 'Ascensión recta'.
etiqueta_ascension = Tkinter.Label(ventana_localizacion,\
text = 'Ascensión recta', font = self.Verd12Roman)
# Colocamos la etiqueta etiqueta_ascension en el
# marco ventana_localizacion.
etiqueta_ascension.grid(row = 2, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)
# Evitamos el cambio de tamaño de la etiqueta etiqueta_ascension.
etiqueta_ascension.grid_propagate(0)

# Creamos una StringVar para mostrar la ascensión recta.
self.text_var_ascensionbuena = Tkinter.StringVar()
# Asignamos un valor nulo a la StringVar creada.
self.text_var_ascensionbuena.set('')

# Creamos una etiqueta para mostrar la ascensión recta usando
# la StringVar anterior.
mostrar_ascension = Tkinter.Label(ventana_localizacion,\
textvariable = self.text_var_ascensionbuena,\
font = self.Verd12Roman)
# Fijamos la posición de la etiqueta creada.
mostrar_ascension.grid(row = 2, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.E)
# Evitamos el cambio de tamaño de la etiqueta anterior.
mostrar_ascension.grid_propagate(0)

# Creamos la etiqueta etiqueta_declinacion y le asignamos
# el texto 'Declinacion'.
etiqueta_declinacion = Tkinter.Label(ventana_localizacion,\
text = 'Declinacion', font = self.Verd12Roman)
# Fijamos la posición de la etiqueta etiqueta_declinacion.
etiqueta_declinacion.grid(row = 3, column = 0, rowspan = 1,\
```

```
columnspan = 1, sticky = Tkinter.W)
# Evitamos la propagación de la etiqueta.
etiqueta_declinacion.grid_propagate(0)

# Creamos el objeto StringVar text_var_declinacionbuena
# para mostrar la declinación.
self.text_var_declinacionbuena = Tkinter.StringVar()
# Ponemos en blanco la StringVar.
self.text_var_declinacionbuena.set('')

# Creamos la etiqueta mostrar_declinacion para visualizar la
# declinacion mediante la StringVar text_var_declinacionbuena.
mostrar_declinacion = Tkinter.Label(ventana_localizacion,\
textvariable = self.text_var_declinacionbuena,\
font = self.Verd12Roman)
# Asignamos una posición a la etiqueta mostrar_declinacion.
mostrar_declinacion.grid(row = 3, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.E)
# Evitamos el cambio de tamaño de la etiqueta anterior.
mostrar_declinacion.grid_propagate(0)

# Creamos una etiqueta con el texto 'Próximo orto:'.
etiqueta_orto = Tkinter.Label(ventana_localizacion,\
text = 'Proximo orto:', font = self.Verd12Roman)
# Colocamos la etiqueta en el marco ventana_localizacion.
etiqueta_orto.grid(row = 4, column = 0, rowspan = 1,\
columnspan = 2, sticky = Tkinter.W)
# Impedimos que esta etiqueta cambie de tamaño.
etiqueta_orto.grid_propagate(0)

# Creamos una StringVar para mostrar el próximo orto del objeto
# elegido.
self.text_var_ortobueno = Tkinter.StringVar()
# Ponemos el texto de la StringVar en blanco.
self.text_var_ortobueno.set('')

# Creamos una etiqueta para mostrar el próximo orto con
# StringVar anterior.
mostrar_orto = Tkinter.Label(ventana_localizacion,\
textvariable = self.text_var_ortobueno, font = self.Verd12Roman)
# Colocamos la etiqueta en el marco ventana_localizacion.
mostrar_orto.grid(row = 5, column = 0, rowspan = 1,\
columnspan = 2, sticky = Tkinter.E)
# Evitamos que la etiqueta cambie de tamaño.
mostrar_orto.grid_propagate(0)

# Creamos una StringVar para mostrar los eventos del sistema.
self.text_var_eventobueno = Tkinter.StringVar()
# Dejamos el texto de la StringVar recién creada en blanco.
self.text_var_eventobueno.set('')

# Creamos una etiqueta para mostrar eventos del sistema utilizando
# la StringVar anterior.
```

```
        mostrar_evento = Tkinter.Label(ventana_localizacion,\
        textvariable = self.text_var_eventobueno, font = self.Verd12Italic)
        # Colocamos la etiqueta en el marco ventana_localizacion.
        mostrar_evento.grid(row = 6, column = 0, rowspan = 1,\
         columnspan = 2)
        # Evitamos que la etiqueta cambie de tamaño.
        mostrar_evento.grid_propagate(0)

    # Si utilizamos un sistema de la familia Windows realizaremos las
    # siguientes rutinas.
    elif sys.platform == "win32":
        # Creamos una etiqueta con el texto 'Altura local'.
        etiqueta_altura = Tkinter.Label(ventana_localizacion,\
        text = 'Altura local', font = self.Verd12Roman)
        # Colocamos la etiqueta en el marco ventana_localizacion.
        etiqueta_altura.grid(row = 0, column = 0, rowspan = 1,\
        columnspan = 2, sticky = Tkinter.W)
        # Evitamos la propagación de la etiqueta.
        etiqueta_altura.grid_propagate(0)

        # Creamos una StringVar para mostrar la altura.
        self.text_var_alturabuena = Tkinter.StringVar()
        # Ponemos en blanco la StringVar creada.
        self.text_var_alturabuena.set('')

        # Creamos la etiqueta mostrar_altura para mostrar la altura
        # mediante la StringVar anterior.
        mostrar_altura = Tkinter.Label(ventana_localizacion,\
        textvariable = self.text_var_alturabuena, font = self.Verd12Roman)
        # Colocamos la etiqueta creada en el marco ventana_localizacion.
        mostrar_altura.grid(row = 0, column = 2, rowspan = 1,\
        columnspan = 1, sticky = Tkinter.E)
        # Evitamos la propagación de la etiqueta.
        mostrar_altura.grid_propagate(0)

        # Creamos la etiqueta etiqueta_acimut con el texto 'Acimut local'.
        etiqueta_acimut = Tkinter.Label(ventana_localizacion,\
        text = 'Acimut local', font = self.Verd12Roman)
        # Fijamos la posición de la etiqueta etiqueta_acimut.
        etiqueta_acimut.grid(row = 1, column = 0, rowspan = 1,\
        columnspan = 2, sticky = Tkinter.W)
        # Evitamos el cambio de tamaño de la etiqueta.
        etiqueta_acimut.grid_propagate(0)

        # Creamos la StringVar text_var_acimutbueno para mostrar el acimut.
        self.text_var_acimutbueno = Tkinter.StringVar()
        # Ponemos en blanco el objeto creado anteriormente.
        self.text_var_acimutbueno.set('')

        # Creamos la etiqueta mostrar_acimut para visualizar el acimut
        # utilizando la StringVar anterior.
        mostrar_acimut = Tkinter.Label(ventana_localizacion,\
        textvariable = self.text_var_acimutbueno, font = self.Verd12Roman)
```

```
# Asignamos una posición en el marco a la etiqueta anterior.
mostrar_acimut.grid(row = 1, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.E)
# Evitamos la propagación de la etiqueta.
mostrar_acimut.grid_propagate(0)

# Creamos una etiqueta con el texto 'Ascensión recta'.
etiqueta_ascension = Tkinter.Label(ventana_localizacion,\
text = 'Ascensión recta', font = self.Verd12Roman)
# Colocamos la etiqueta etiqueta_ascension en el
# marco ventana_localizacion.
etiqueta_ascension.grid(row = 2, column = 0, rowspan = 1,\
columnspan = 2, sticky = Tkinter.W)
# Evitamos el cambio de tamaño de la etiqueta etiqueta_ascension.
etiqueta_ascension.grid_propagate(0)

# Creamos una StringVar para mostrar la ascensión recta.
self.text_var_ascensionbuena = Tkinter.StringVar()
# Asignamos un valor nulo a la StringVar creada.
self.text_var_ascensionbuena.set('')

# Creamos una etiqueta para mostrar la ascensión recta usando
# la StringVar anterior.
mostrar_ascension = Tkinter.Label(ventana_localizacion,\
textvariable = self.text_var_ascensionbuena,\
font = self.Verd12Roman)
# Fijamos la posición de la etiqueta creada.
mostrar_ascension.grid(row = 2, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.E)
# Evitamos el cambio de tamaño de la etiqueta anterior.
mostrar_ascension.grid_propagate(0)

# Creamos la etiqueta etiqueta_declinacion y le asignamos
# el texto 'Declinacion'.
etiqueta_declinacion = Tkinter.Label(ventana_localizacion,\
text = 'Declinacion', font = self.Verd12Roman)
# Fijamos la posición de la etiqueta etiqueta_declinacion.
etiqueta_declinacion.grid(row = 3, column = 0, rowspan = 1,\
columnspan = 2, sticky = Tkinter.W)
# Evitamos la propagación de la etiqueta.
etiqueta_declinacion.grid_propagate(0)

# Creamos la etiqueta text_var_declinacionbuena
# para mostrar la declinación.
self.text_var_declinacionbuena = Tkinter.StringVar()
# Ponemos en blanco la StringVar.
self.text_var_declinacionbuena.set('')

# Creamos la etiqueta mostrar_declinacion para visualizar la
# declinacion mediante la StringVar text_var_declinacionbuena.
mostrar_declinacion = Tkinter.Label(ventana_localizacion,\
textvariable = self.text_var_declinacionbuena,\
font = self.Verd12Roman)
```

```
        # Asignamos una posición a la etiqueta mostrar_declinacion
        mostrar_declinacion.grid(row = 3, column = 2, rowspan = 1,\
        columnspan = 1, sticky = Tkinter.E)
        # Evitamos el cambio de tamaño de la etiqueta anterior.
        mostrar_declinacion.grid_propagate(0)

        # Creamos la etiqueta etiqueta_satelite con el texto 'Satélite:'.
        etiqueta_satelite = Tkinter.Label(ventana_localizacion,\
        text = 'Satélite:', font = self.Verd12Roman)
        # Asignamos una posición a la etiqueta anterior.
        etiqueta_satelite.grid(row = 4, column = 0, rowspan = 1,
                               columnspan = 1, sticky = Tkinter.W)
        # Evitamos el cambio de tamaño de la etiqueta anterior.
        etiqueta_localizacion.grid_propagate(0)

        # Creacion de la StringVar text_var_satelitebueno para mostrar
        # el nombre del satélite.
        self.text_var_satelitebueno = Tkinter.StringVar()
        # Ponemos en blanco el texto del objeto anterior.
        self.text_var_satelitebueno.set('')

        # Creamos una etiqueta para mostrar el nombre del satélite mediante
        # la StringVar anterior.
        mostrar_satelite = Tkinter.Label(ventana_localizacion,\
        textvariable = self.text_var_satelitebueno,\
        font = self.Verd12Italic)
        # Colocamos la etiqueta creada en el marco ventana_localizacion.
        mostrar_satelite.grid(row = 4, column = 1, rowspan = 1,
                              columnspan = 2, sticky = Tkinter.E)
        # Evitamos el cambio de tamaño de mostrar_satelite.
        mostrar_satelite.grid_propagate(0)

        # Creamos una StringVar para mostrar los eventos del programa.
        self.text_var_eventobueno = Tkinter.StringVar()
        # Ponemos en blanco el texto de la StringVar.
        self.text_var_eventobueno.set('')

        # Creamos una etiqueta para mostrar los eventos del programa
        # mediante la StringVar creada anteriormente.
        mostrar_evento = Tkinter.Label(ventana_localizacion,\
        textvariable = self.text_var_eventobueno, font = self.Verd12Roman)
        # Asignamos una posición a la etiqueta en su marco.
        mostrar_evento.grid(row = 5, column = 0, rowspan = 1,
                            columnspan = 3)
        # Evitamos el cambio de tamaño de la etiqueta.
        mostrar_evento.grid_propagate(0)

    ##########################################################################

    # A continuación crearemos un marco para contener las dos listas
    # necesarias para la selección de un objeto.
    # Una es la lista con las familias disponibles y otra es la lista con
    # los satélites de cada familia.
```

```
ventana_seleccion = Tkinter.LabelFrame(root,\
text = 'Selección del satelite', font = self.Verd12Italic,\
width = 190, height = 167)
# Colocaremos el marco en la ventana principal.
ventana_seleccion.grid(row = 1, column = 2, rowspan = 1,\
columnspan = 1, padx=5, pady = 5, sticky = Tkinter.N)
# Evitamos la propagación del marco por la ventana principal.
ventana_seleccion.grid_propagate(0)
# Asignamos un tamaño mínimo de 50 píxeles a la primera columna.
ventana_seleccion.columnconfigure(0, minsize = 50)
# Asignamos un tamaño mínimo de 90 píxeles a la segunda columna.
ventana_seleccion.columnconfigure(1, minsize = 90)
# Fijamos un tamaño mínimo de 50 píxeles a la primera fila.
ventana_seleccion.rowconfigure(0, minsize = 50)
# Fijamos un tamaño mínimo de 50 píxeles a la segunda fila.
ventana_seleccion.rowconfigure(1, minsize = 50)

# Sistemas de la familia Macintosh.
if sys.platform == "darwin":
    # Creamos un objeto de tipo ScrolledList que es un menú
    # despegable. Vinculamos este objeto a la función
    # seleccionar_familia necesaria para la elección de una familia.
    self.lista_tipossatelite = scrolledlist.ScrolledList\
    (ventana_seleccion, width = 19, height = 3,\
     callback = self.seleccionar_familia)
    # Colocamos el menú en el marco ventana_seleccion.
    self.lista_tipossatelite.grid (row = 0, column = 0, rowspan = 1,\
    columnspan = 2, sticky = Tkinter.W, padx = 5, pady = 2)

    # Generamos otro objeto del tipo ScrolledList. Este menú será
    # necesario para la selección de un satélite y estará vinculado a
    # la función seleccionar_satelite.
    self.lista_satelites = scrolledlist.ScrolledList\
    (ventana_seleccion, width = 19, height = 3,\
     callback = self.seleccionar_satelite)
    # Colocamos el menú creado en el marco ventana_seleccion.
    self.lista_satelites.grid (row = 1, column = 0, rowspan = 1,\
    columnspan = 2, sticky = Tkinter.W, padx = 5)

# Sistemas operativos basados en Linux.
elif sys.platform.startswith('linux'):
    # Creamos un objeto de tipo ScrolledList que es un menú
    # despegable. Vinculamos este objeto a la función
    # seleccionar_familia necesaria para la elección de una familia.
    self.lista_tipossatelite = scrolledlist.ScrolledList\
    (ventana_seleccion, width = 19, height = 3,\
     callback = self.seleccionar_familia)
    # Colocamos el menú en el marco ventana_seleccion.
    self.lista_tipossatelite.grid (row = 0, column = 0, rowspan = 1,\
    columnspan = 2, sticky = Tkinter.W, padx = 5, pady = 2)

    # Generamos otro objeto del tipo ScrolledList. Este menú será
    # necesario para la selección de un satélite y estará vinculado a
```

```
        # la función seleccionar_satelite.
        self.lista_satelites = scrolledlist.ScrolledList\
        (ventana_seleccion, width = 19, height = 4,\
         callback = self.seleccionar_satelite)
        # Colocamos el menú creado en el marco ventana_seleccion.
        self.lista_satelites.grid (row = 1, column = 0, rowspan = 1,\
        columnspan = 2, sticky = Tkinter.W, padx = 5)

    # Sistemas operativos de la familia Windows.
    elif sys.platform == "win32":
        # Creamos un objeto de tipo ScrolledList que es un menú
        # despegable. Vinculamos este objeto a la función
        # seleccionar_familia necesaria para la elección de una familia.
        self.lista_tipossatelite = scrolledlist.ScrolledList\
        (ventana_seleccion, width = 21, height = 3,\
         callback = self.seleccionar_familia)
        # Definimos el tipo de letra de los objetos del menú para
        # así ajustar su tamaño al espacio disponible.
        self.lista_tipossatelite.listbox.configure(font = self.Verd12Roman)
        # Colocamos el menú en el marco ventana_seleccion.
        self.lista_tipossatelite.grid (row = 0, column = 0, rowspan = 1,\
        columnspan = 2, sticky = Tkinter.W, padx = 5, pady = 2)

        # Generamos otro objeto del tipo ScrolledList. Este menú será
        # necesario para la selección de un satélite y estará vinculado a
        # la función seleccionar_satelite.
        self.lista_satelites = scrolledlist.ScrolledList\
        (ventana_seleccion, width = 21, height = 4,\
         callback = self.seleccionar_satelite)
        # Definimos el tipo de letra de los objetos del menú para
        # así ajustar su tamaño al espacio disponible.
        self.lista_satelites.listbox.configure(font = self.Verd12Roman)
        # Colocamos el menú creado en el marco ventana_seleccion.
        self.lista_satelites.grid (row = 1, column = 0, rowspan = 1,\
        columnspan = 2, sticky = Tkinter.W, padx = 5)

    # A continuación generaremos una serie de widgets para mostrar el
    # satélite seleccionado.
    # Debido a restricciones de tamaño estos solo podrán mostrarse en
    # Mac OS X (en sus diferentes versiones) y en las distribuciones de
    # Linux.
    # Sistemas operativos de la familia Macintosh.
    if sys.platform == 'darwin':
        # Generamos una etiqueta con el texto 'Satélite:'.
        etiqueta_satelite = Tkinter.Label(ventana_seleccion,\
        text = 'Satélite:', font = self.Verd12Italic)
        # Asignamos una posición a la etiqueta etiqueta_satelite en el
        # marco ventana_selección.
        etiqueta_satelite.grid(row = 2, column = 0, rowspan = 1,\
        columnspan = 1, sticky = Tkinter.W)
        # Impedimos el cambio de tamaño de la etiqueta.
        etiqueta_localizacion.grid_propagate(0)
```

```
        # Creamos un objeto del tipo StringVar para mostrar el nombre
        # del satélite.
        self.text_var_satelitebueno = Tkinter.StringVar()
        # Ponemos en blanco el texto del objeto.
        self.text_var_satelitebueno.set('')

        # Generamos una etiqueta para mostrar el nombre del satélite
        # modificando el texto de la StringVar anterior.
        mostrar_satelite = Tkinter.Label(ventana_seleccion,\
        textvariable = self.text_var_satelitebueno,\
        font = self.Verd12Italic)
        # Colocamos la etiqueta mostrar_satelite en el marco.
        mostrar_satelite.grid(row = 2, column = 1, rowspan = 1,\
        columnspan = 1, sticky = Tkinter.E)
        # Impedimos el cambio de tamaño de la etiqueta.
        mostrar_satelite.grid_propagate(0)

    # Sistemas operativos basados en Linux.
    elif sys.platform.startswith('linux'):
        # Generamos una etiqueta con el texto 'Satélite:'.
        etiqueta_satelite = Tkinter.Label(ventana_seleccion,\
        text = 'Satélite:', font = self.Verd12Italic)
        # Asignamos una posición a la etiqueta etiqueta_satelite en el
        # marco ventana_selección.
        etiqueta_satelite.grid(row = 2, column = 0, rowspan = 1,\
        columnspan = 1, sticky = Tkinter.W)
        # Impedimos el cambio de tamaño de la etiqueta.
        etiqueta_localizacion.grid_propagate(0)

        # Creamos un objeto del tipo StringVar para mostrar el nombre
        # del satélite.
        self.text_var_satelitebueno = Tkinter.StringVar()
        # Ponemos en blanco el texto del objeto.
        self.text_var_satelitebueno.set('')

        # Generamos una etiqueta para mostrar el nombre del satélite
        # modificando el texto de la StringVar anterior.
        mostrar_satelite = Tkinter.Label(ventana_seleccion,\
        textvariable = self.text_var_satelitebueno,\
        font = self.Verd12Italic)
        # Colocamos la etiqueta mostrar_satelite en el marco.
        mostrar_satelite.grid(row = 2, column = 1, rowspan = 1,\
        columnspan = 1, sticky = Tkinter.E)
        # Impedimos el cambio de tamaño de la etiqueta.
        mostrar_satelite.grid_propagate(0)

    ###########################################################################

    # Generaremos una nueva ventana llamada ventana_controles donde
    # colocaremos un widget del tipo Notebook que contendrá los controles
    # para la selección de un tipo de conexión y el cambio de velocidad de
    # actualización del sistema.
    ventana_controles = Tkinter.LabelFrame(root,\
```

```
text = 'Controles del sistema', font = self.Verd12Italic,\
width = 390, height = 159)
# Fijamos una posición para el marco en la ventana root.
ventana_controles.grid(row = 2, column = 1, rowspan = 2,\
columnspan = 2, padx = 5, pady = 5)
# Evitamos el cambio de tamaño del Notebook dentro de root.
ventana_controles.grid_propagate(0)

# El wigdet Notebook posee diferentes esquemas de diseño dependiendo
# del sistema operativo utilizado. Esto nos obligará a fijar un tamaño
# de ventanas y columnas diferente en cada uno de ellos.
# Sistemas operativos de la serie Mac OS X.
if sys.platform == "darwin":
    # Ancho de la primera columna en píxeles.
    ventana_controles.columnconfigure(0, minsize = 270)
    # Ancho de la segunda columna en píxeles.
    ventana_controles.columnconfigure(1, minsize = 40)

# Sistemas operativos basados en el kernel de Linux.
elif sys.platform.startswith('linux'):
    # Ancho de la primera columna en píxeles.
    ventana_controles.columnconfigure(0, minsize = 270)
    # Ancho de la segunda columna en píxeles.
    ventana_controles.columnconfigure(1, minsize = 40)

# Sistemas operativos de la familia Windows.
elif sys.platform == "win32":
    # Ancho de la primera columna en píxeles.
    ventana_controles.columnconfigure(0, minsize = 280)
    # Ancho de la segunda columna en píxeles.
    ventana_controles.columnconfigure(1, minsize = 40)

# Creamos un objeto de tipo Notebook para contener los controles.
# Sistemas operativos de la familia Macintosh.
if sys.platform == "darwin":
    # Creamos un objeto de tipo Notebook en el marco ventana_controles.
    estados_notebook = ttk.Notebook(ventana_controles, height = 80,
                                    width = 205)
    # Asignamos una posición al objeto en el marco.
    estados_notebook.grid(row = 0, column = 0, rowspan = 5,
                          columnspan = 1)
    # Evitamos el cambio de tamaño del objeto.
    estados_notebook.grid_propagate(0)

# Distribuciones basadas en Linux.
elif sys.platform.startswith('linux'):
    # Creamos un objeto de tipo Notebook en el marco ventana_controles.
    estados_notebook = ttk.Notebook(ventana_controles, height = 110,
                                    width = 250)
    # Asignamos una posición al objeto en el marco.
    estados_notebook.grid(row = 0, column = 0, rowspan = 5,
                          columnspan = 1)
    # Evitamos el cambio de tamaño del objeto.
```

```
    estados_notebook.grid_propagate(0)

# Sistemas operativos de la familia Windows.
elif sys.platform == "win32":
    # Creamos un objeto de tipo Notebook en el marco ventana_controles.
    estados_notebook = ttk.Notebook(ventana_controles, height = 110,
                                    width = 260)
    # Asignamos una posición al objeto en el marco.
    estados_notebook.grid(row = 0, column = 0, rowspan = 5,
                          columnspan = 1)
    # Evitamos el cambio de tamaño del objeto
    estados_notebook.grid_propagate(0)

# Generamos la primera pagina del Notebook.
primera_ventana = ttk.Frame(estados_notebook)

# A continuación fijamos los tamaños mínimos de las filas y las
# columnas de la página.
# Sistemas operativos de la serie Mac OS X.
if sys.platform == "darwin":
    # Asignamos un tamaño mínimo de 25 píxeles a la primera fila.
    primera_ventana.rowconfigure(0, minsize = 25)
    # Asignamos un tamaño mínimo de 40 píxeles a la segunda fila.
    primera_ventana.rowconfigure(1, minsize = 25)
    # Asignamos un tamaño mínimo de 110 píxeles a la primera columna.
    primera_ventana.columnconfigure(0, minsize = 110)

# Sistemas operativos basados en Linux.
elif sys.platform.startswith('linux'):
    # Fijamos un tamaño mínimo de 25 píxeles a la primera fila.
    primera_ventana.rowconfigure(0, minsize = 25)
    # Fijamos un tamaño mínimo de 25 píxeles a la segunda fila.
    primera_ventana.rowconfigure(1, minsize = 25)
    # Fijamos un tamaño mínimo de 25 píxeles a la tercera fila.
    primera_ventana.rowconfigure(2, minsize = 25)
    # Fijamos un tamaño mínimo de 25 píxeles a la cuarta fila.
    primera_ventana.rowconfigure(3, minsize = 25)
    # Fijamos un tamaño mínimo de 130 píxeles a la primera columna.
    primera_ventana.columnconfigure(0, minsize = 130)

# Sistemas operativos de la familia Windows.
elif sys.platform == "win32":
    # Asignamos un tamaño mínimo a la primera fila de 25 píxeles.
    primera_ventana.rowconfigure(0, minsize = 25)
    # Asignamos un tamaño mínimo a la segunda fila de 25 píxeles.
    primera_ventana.rowconfigure(1, minsize = 25)
    # Asignamos un tamaño mínimo a la tercera fila de 25 píxeles.
    primera_ventana.rowconfigure(2, minsize = 25)
    # Asignamos un tamaño mínimo a la cuarta fila de 25 píxeles.
    primera_ventana.rowconfigure(3, minsize = 25)
    # Asignamos un tamaño mínimo a la primera columna de 155 píxeles.
    primera_ventana.columnconfigure(0, minsize = 155)
```

```
# Añadimos la primera página al widget Notebook primera_ventana.
estados_notebook.add(primera_ventana, text='Conexión')

# Creamos una etiqueta para señalar el widget de selección de
# configuraciones.
# Sistemas operativos de la serie Mac OS X.
if sys.platform == "darwin":
    # Generamos la etiqueta etiqueta_configuracion con el texto
    # 'Protocolo'.
    etiqueta_configuracion = Tkinter.Label(primera_ventana,\
    text = 'Protocolo:', font = self.Verd11Roman)
    # Asignamos una posición a la etiqueta creada en la primera pagina.
    etiqueta_configuracion.grid(row = 0, column = 0, rowspan = 1,\
    columnspan = 1, sticky = Tkinter.W)

# Sistemas operativos basados en el kernel de Linux.
elif sys.platform.startswith('linux'):
    # Generamos la etiqueta etiqueta_configuracion con el texto
    # 'Configuración:'.
    etiqueta_configuracion = Tkinter.Label(primera_ventana,\
    text = 'Configuración:', font = self.Verd12Roman)
    # Asignamos una posición a la etiqueta creada en la primera pagina.
    etiqueta_configuracion.grid(row = 0, column = 0, rowspan = 1,\
    columnspan = 1, sticky = Tkinter.W)

# Sistemas operativos de la familia Windows.
elif sys.platform == "win32":
    # Generamos la etiqueta etiqueta_configuracion con el texto
    # 'Configuración:'.
    etiqueta_configuracion = Tkinter.Label(primera_ventana,\
    text = 'Configuración:', font = self.Verd11Roman)
    # Asignamos una posición a la etiqueta creada en la primera pagina.
    etiqueta_configuracion.grid(row = 0, column = 0, rowspan = 1,\
    columnspan = 1, sticky = Tkinter.W)

# Creamos una lista vacia para almacenar un listado de nombres
# de configuraciones.
self.lista_configuraciones = []

# Abrimos el archivo con las configuraciones y almacenamos
# su valor en listar_configuraciones.
listar_configuraciones = csv.reader(open("confconexion.list", 'rb'))

# Leemos la primera columna del archivo listar_configuraciones
for row in listar_configuraciones:
    # Añadimos el primer valor de cada fila a la
    # lista self.lista_configuraciones.
    self.lista_configuraciones.append(row[0])

# Creamos un objeto del tipo Spinbox para la selección de las
# configuraciones disponibles.
# Este objeto creará un menú. Al seleccionar un elemento de dicho
# menú se invocará la función que deseemos. En este caso
```

```
# self.fijar_valores_conexion.
# Le damos como posibles valores al menú los elementos de
# la lista lista_configuraciones.
self.spinbox_configuraciones = Tkinter.Spinbox(primera_ventana,\
font = self.Verd11Roman, values = self.lista_configuraciones,\
width = 9, command = self.fijar_valores_conexion)
# Colocamos la Spinbox creada en su posición de la primera pagina.
self.spinbox_configuraciones.grid(row = 0, column = 1,\
rowspan = 1, columnspan = 1)

# Sistemas operativos de la familia Macintosh.
if sys.platform == "darwin":
    # Crearemos un botón para establecer una conexión con los rotores.
    # Lo vincularemos a la función establecer_conexión.
    boton_conexion = Tkinter.Button(primera_ventana,\
    text = 'Fijar conexión', font = self.Verd11Roman, width = 12,\
    command = self.establecer_conexion)
    # Asignamos una posición al botón en la primera ventana.
    boton_conexion.grid(row = 1, column = 0, rowspan = 1,\
    columnspan = 2, sticky = Tkinter.E)

# Distribuciones basadas en el kernel de Linux.
elif sys.platform.startswith('linux'):
    # Crearemos un botón para establecer una conexión con los rotores.
    # Lo vincularemos a la función establecer_conexión.
    boton_conexion = Tkinter.Button(primera_ventana,\
    text = 'Establecer conexión', font = self.Verd11Roman, width = 18,\
    command = self.establecer_conexion)
    # Asignamos una posición al botón en la primera ventana.
    boton_conexion.grid(row = 1, column = 0, rowspan = 1,\
    columnspan = 2, sticky = Tkinter.E)

# Sistemas operativos de la serie Windows.
elif sys.platform == "win32":
    # Crearemos un botón para establecer una conexión con los rotores.
    # Lo vincularemos a la función establecer_conexión.
    boton_conexion = Tkinter.Button(primera_ventana,\
    text = 'Establecer conexión', font = self.Verd11Roman, width = 18,\
    command = self.establecer_conexion)
    # Fijamos una posición para el botón en la primera ventana.
    boton_conexion.grid(row = 1, column = 0, rowspan = 1,\
    columnspan = 2, sticky = Tkinter.E)

# Generamos un objeto del tipo StringVar para mostrar avisos relativos
# a las conexiones del sistema o al estado de los rotores.
self.text_var_mostrarestado = Tkinter.StringVar()
# Asignamos un texto de referencia al objeto.
self.text_var_mostrarestado.set('Avisos del sistema')

# Creamos una etiqueta llamada mostrar_estado_conexion para, utilizando
# el StringVar anterior mostrar las advertencias.
# Sistemas operativos de la serie Mac OS X.
if sys.platform == "darwin":
```

```
        # Creamos la etiqueta.
        mostrar_estado_conexion = Tkinter.Label(primera_ventana,\
        textvariable = self.text_var_mostrarestado,\
        font = self.Verd11Roman)
        # Colocamos la etiqueta en la primera página.
        mostrar_estado_conexion.grid(row = 2, column = 0,\
        rowspan = 1, columnspan = 2)

    # Sistemas operativos basados en Linux.
    elif sys.platform.startswith('linux'):
        # Creamos la etiqueta.
        mostrar_estado_conexion = Tkinter.Label(primera_ventana,\
        textvariable = self.text_var_mostrarestado,\
        font = self.Verd12Italic)
        # Colocamos la etiqueta en la primera página.
        mostrar_estado_conexion.grid(row = 2, column = 0,\
        rowspan = 1, columnspan = 2)

    # Sistemas operativos de la serie Windows.
    elif sys.platform == "win32":
        # Creamos la etiqueta.
        mostrar_estado_conexion = Tkinter.Label(primera_ventana,\
        textvariable = self.text_var_mostrarestado,\
        font = self.Verd11Roman)
        # Colocamos la etiqueta en la primera página.
        mostrar_estado_conexion.grid(row = 2, column = 0,\
        rowspan = 1, columnspan = 2)

    # Creamos un botón para abrir la ventana de configuración del sistema.
    # Debido a las diferencias existentes en el diseño de los temas de
    # Tkinter para cada sistema operativo este botón se encontrará en una
    # posición distinta en cada caso.
    # Mostraremos las particularidades de cada caso.
    # Sistemas operativos de la serie Mac OS X.
    if sys.platform == "darwin":
        # Creamos el botón boton_configuracion vinculándolo a la función
        # configurar_sistema. Colocamos el botón en el marco
        # ventana_controles.
        self.boton_configuracion = Tkinter.Button(ventana_controles,\
        text = 'Configuración', font = self.Verd12Roman, height = 1,\
        width = 11, command = self.configurar_sistema)
        # Le asignamos una posición al botón en el marco.
        self.boton_configuracion.grid(row = 3, column = 1)
        # Fijamos un nuevo valor a row_nuevo.
        row_nuevo = 3

    # Sistemas operativos basados en Linux.
    elif sys.platform.startswith('linux'):
        # Creamos el botón boton_configuracion vinculándolo a la función
        # configurar_sistema. Colocamos el botón en la primera página.
        self.boton_configuracion = Tkinter.Button(primera_ventana,\
        text = 'Configuración', font = self.Verd12Roman, height = 1,\
        width = 11, command = self.configurar_sistema)
```

```
        # Fijamos la posición del botón en la primera página.
        self.boton_configuracion.grid(row = 3, column = 1,\
        rowspan = 1, columnspan = 1)
        # Asignamos un valor a row_nuevo.
        row_nuevo = 2

    # Sistemas operativos de la serie Windows.
    elif sys.platform == "win32":
        # Creamos el botón boton_configuracion vinculándolo a la función
        # configurar_sistema. Colocamos el botón en la primera página.
        self.boton_configuracion = Tkinter.Button(primera_ventana,\
        text = 'Configuración', font = self.Verd12Roman, height = 1,\
        width = 11, command = self.configurar_sistema)
        # Fijamos la posición del botón en la primera página.
        self.boton_configuracion.grid(row = 3, column = 1,\
        rowspan = 1, columnspan = 1)
        # Asignamos un valor a row_nuevo.
        row_nuevo = 2

    # Creamos la segunda página del objeto Notebook.
    segunda_ventana = ttk.Frame(estados_notebook)
    # La añadimos al objeto Notebook y le damos el nombre 'Seguimiento'.
    estados_notebook.add(segunda_ventana, text='Seguimiento')

    # Creamos los elementos de la segunda página.
    # Usaremos una sentencia if-elif para cada sistema operativo.
    # Sistemas operativos de la familia Macintosh.
    if sys.platform == "darwin":
        # Generamos una etiqueta con el texto 'Coordenadas:'.
        etiqueta_velocidad = Tkinter.Label(segunda_ventana,\
        text = 'Coordenadas:', font = self.Verd11Roman)
        # Asignamos una posición a la etiqueta en la página.
        etiqueta_velocidad.grid(row = 0, column = 0, rowspan = 1,\
        columnspan = 1, sticky = Tkinter.W)

        # Creamos un campo de entrada de texto para introducir un valor
        # (en segundos) de actualización de los valores de las coordenadas.
        self.posibles_actualizaciones_coordenadas = Tkinter.Entry\
        (segunda_ventana, width = 2, font = self.Verd11Roman)
        # Colocamos el campo de entrada de texto en la página.
        self.posibles_actualizaciones_coordenadas.grid(row = 0,\
        column = 2, rowspan = 1, columnspan = 1)

        # Creamos una etiqueta con el texto 'Rotores:'.
        etiqueta_rotores = Tkinter.Label(segunda_ventana,\
        text = 'Rotores:', font = self.Verd11Roman)
        # Colocamos la etiqueta en su posición de la segunda página.
        etiqueta_rotores.grid(row = 1, column = 0, rowspan = 1,\
        columnspan = 1, sticky = Tkinter.W)

        # Creamos otro campo de entrada de texto para asignar un tiempo
        # mínimo de actualización de la posición de los rotores.
        self.posibles_actualizaciones_rotores = Tkinter.Entry\
```

```
        (segunda_ventana, width = 2, font = self.Verd11Roman)
        # Asignamos una posición al campo de entrada en la página.
        self.posibles_actualizaciones_rotores.grid(row = 1, column = 2,\
        rowspan = 1, columnspan = 1)

        # Creamos una etiqueta llamada etiqueta_automatica con el texto
        # 'Modo automático:'.
        etiqueta_automatica = Tkinter.Label(segunda_ventana,\
        text = 'Modo automático:', font = self.Verd11Roman)
        # Fijamos la posición de la etiqueta anterior en la segunda página.
        etiqueta_automatica.grid(row = 2, column = 0, rowspan = 1,\
                                 columnspan = 1, sticky = Tkinter.W)

        # Creamos un StringVar para almacenar el modo de actualización
        # del sistema mediante una cadena de texto.
        self.control_actualizacion = Tkinter.StringVar()

        # A continuación generamos un nuevo objeto del tipo Radiobutton.
        # Este objeto nos permitirá elegir una opción entre varias
        # presionando un boton. La novedad de este widget es que solo
        # permite una elección al mismo tiempo.
        # Creamos la primera elección, que será 'Si'.
        # Al presionar el botón este objeto dará el valor 'Si' a la función
        # fijar_modo_actualización.
        eleccion_Si = Tkinter.Radiobutton(segunda_ventana, text = 'Si',\
        font = self.Verd11Roman, command = self.fijar_modo_actualizacion,\
        variable = self.control_actualizacion, value = 'Si')
        # Colocamos el botón en la segunda página.
        eleccion_Si.grid(row = 2, column = 1, rowspan = 1, columnspan = 1)
        # Ahora nos tocará la siguiente opción, que es 'No'
        # Al presionar el botón este objeto dará el valor 'No' a la función
        # fijar_modo_actualización.
        eleccion_No = Tkinter.Radiobutton(segunda_ventana, text = 'No',\
        font = self.Verd11Roman, command = self.fijar_modo_actualizacion,\
        variable = self.control_actualizacion, value = 'No')
        # Colocamos el botón en la segunda página.
        eleccion_No.grid(row = 2, column = 2, rowspan = 1, columnspan = 1)

    # Sistemas operativos basados en el kernel de Linux.
    elif sys.platform.startswith('linux'):
        # Generamos la etiqueta etiqueta_manual con el texto
        # 'Intervalos de actualización en segundos'.
        etiqueta_manual = Tkinter.Label(segunda_ventana,\
        text = 'Intervalos de actualización en segundos:',\
        font = self.Verd11Roman)
        # Fijamos la posición de la etiqueta en la página.
        etiqueta_manual.grid(row = 0, column = 0,
                             rowspan = 1, columnspan = 3)

        # Creamos una etiqueta con el texto 'Actualización coordenadas'.
        etiqueta_velocidad = Tkinter.Label(segunda_ventana,\
        text = 'Actualización coordenadas:', font = self.Verd11Roman)
        # Asignamos una posición a la etiqueta en la página.
```

```
etiqueta_velocidad.grid(row = 1, column = 0, rowspan = 1,
                        columnspan = 1, sticky = Tkinter.W)

# Creamos un campo de entrada de texto para introducir un valor
# (en segundos) de actualización de los valores de las coordenadas.
self.posibles_actualizaciones_coordenadas = Tkinter.Entry\
(segunda_ventana, width = 2, font = self.Verd11Roman)
# Colocamos el campo de entrada de texto en la página.
self.posibles_actualizaciones_coordenadas.grid(row = 1,\
column = 2, rowspan = 1, columnspan = 1)

# Creamos una etiqueta con el texto 'Movimiento rotores:'.
etiqueta_rotores = Tkinter.Label(segunda_ventana,\
text = 'Movimiento rotores:', font = self.Verd11Roman)
# Colocamos la etiqueta en su posición de la segunda página.
etiqueta_rotores.grid(row = 2, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos otro campo de entrada de texto para asignar un tiempo
# mínimo de actualización de la posición de los rotores.
self.posibles_actualizaciones_rotores = Tkinter.Entry\
(segunda_ventana, width = 2, font = self.Verd11Roman)
# Asignamos una posición al campo de entrada en la página.
self.posibles_actualizaciones_rotores.grid(row = 2, column = 2,\
rowspan = 1, columnspan = 1)

# Creamos una etiqueta llamada etiqueta_automatica con el texto
# 'Actualización automática:'.
etiqueta_automatica = Tkinter.Label(segunda_ventana,\
text = 'Actualización automática:', font = self.Verd11Roman)
# Fijamos la posición de la etiqueta anterior en la segunda página.
etiqueta_automatica.grid(row = 3, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un StringVar para almacenar el modo de actualización
# del sistema mediante una cadena de texto.
self.control_actualizacion = Tkinter.StringVar()

# A continuación generamos un nuevo objeto del tipo Radiobutton.
# Este objeto nos permitirá elegir una opción entre varias
# presionando un boton. La novedad de este widget es que solo
# permite una elección al mismo tiempo.
# Creamos la primera elección, que será 'Si'.
# Al presionar el botón este objeto dará el valor 'Si' a la función
# fijar_modo_actualización.
eleccion_Si = Tkinter.Radiobutton(segunda_ventana, text = 'Si',\
font = self.Verd11Roman, command = self.fijar_modo_actualizacion,\
variable = self.control_actualizacion, value = 'Si')
# Colocamos el botón en su posición de la segunda página.
eleccion_Si.grid(row = 3, column = 1, rowspan = 1, columnspan = 1)
# Ahora nos tocará la siguiente opción, que es 'No'
# Al presionar el botón este objeto dará el valor 'No' a la función
# fijar_modo_actualización.
```

```
        eleccion_No = Tkinter.Radiobutton(segunda_ventana, text = 'No',\
        font = self.Verd11Roman, command = self.fijar_modo_actualizacion,\
        variable = self.control_actualizacion, value = 'No')
        # Colocamos el botón en la segunda página.
        eleccion_No.grid(row = 3, column = 2, rowspan = 1, columnspan = 1)

    # Sistemas operativos de la familia Windows.
    elif sys.platform == "win32":
        # Creamos una etiqueta con el texto 'Tiempos de actualización en
        # segundos:'.
        etiqueta_manual = Tkinter.Label(segunda_ventana,\
        text = 'Tiempos de actualización en segundos:',\
        font = self.Verd11Roman)
        # Asignamos una posición a la etiqueta en la segunda página.
        etiqueta_manual.grid(row = 0, column = 0,
                             rowspan = 1, columnspan = 3)

        # Generamos una etiqueta con el texto 'Coordenadas:'.
        etiqueta_velocidad = Tkinter.Label(segunda_ventana,\
        text = 'Coordenadas:', font = self.Verd11Roman)
        # Asignamos una posición a la etiqueta en la página.
        etiqueta_velocidad.grid(row = 1, column = 0, rowspan = 1,\
        columnspan = 1, sticky = Tkinter.W)

        # Creamos un campo de entrada de texto para introducir un valor
        # (en segundos) de actualización de los valores de las coordenadas.
        self.posibles_actualizaciones_coordenadas = Tkinter.Entry\
        (segunda_ventana, width = 2, font = self.Verd11Roman)
        # Colocamos el campo de entrada de texto en la página.
        self.posibles_actualizaciones_coordenadas.grid(row = 1, column = 2,\
        rowspan = 1, columnspan = 1)

        # Creamos una etiqueta con el texto 'Rotores:'.
        etiqueta_rotores = Tkinter.Label(segunda_ventana,\
        text = 'Rotores:', font = self.Verd11Roman)
        # Colocamos la etiqueta en su posición de la segunda página.
        etiqueta_rotores.grid(row = 2, column = 0, rowspan = 1,\
        columnspan = 1, sticky = Tkinter.W)

        # Creamos otro campo de entrada de texto para asignar un tiempo
        # mínimo de actualización de la posición de los rotores.
        self.posibles_actualizaciones_rotores = Tkinter.Entry\
        (segunda_ventana, width = 2, font = self.Verd11Roman)
        # Asignamos una posición al campo de entrada en la página.
        self.posibles_actualizaciones_rotores.grid(row = 2, column = 2,\
        rowspan = 1, columnspan = 1)

        # Creamos una etiqueta llamada etiqueta_automatica con el texto
        # 'Modo automático:'.
        etiqueta_automatica = Tkinter.Label(segunda_ventana,\
        text = 'Modo automático:', font = self.Verd11Roman)
        # Fijamos la posición de la etiqueta anterior en la segunda página.
        etiqueta_automatica.grid(row = 3, column = 0, rowspan = 1,\
```

```
columnspan = 1, sticky = Tkinter.W)

# Creamos una StringVar para almacenar el modo de actualización
# del sistema mediante una cadena de texto.
self.control_actualizacion = Tkinter.StringVar()

# A continuación generamos un nuevo objeto del tipo Radiobutton.
# Este objeto nos permitirá elegir una opción entre varias
# presionando un boton. La novedad de este widget es que solo
# permite una elección al mismo tiempo.
# Creamos la primera elección, que será 'Si'.
# Al presionar el botón este objeto dará el valor 'Si' a la función
# fijar_modo_actualización.
eleccion_Si = Tkinter.Radiobutton(segunda_ventana, text = 'Si',\
font = self.Verd11Roman, command = self.fijar_modo_actualizacion,\
variable = self.control_actualizacion, value = 'Si')
# Colocamos el botón en la segunda página.
eleccion_Si.grid(row = 3, column = 1, rowspan = 1, columnspan = 1)
# Ahora nos tocará la siguiente opción, que es 'No'
# Al presionar el botón este objeto dará el valor 'No' a la función
# fijar_modo_actualización.
eleccion_No = Tkinter.Radiobutton(segunda_ventana, text = 'No',\
font = self.Verd11Roman, command = self.fijar_modo_actualizacion,\
variable = self.control_actualizacion, value = 'No')
# Colocamos el botón en la segunda página.
eleccion_No.grid(row = 3, column = 2, rowspan = 1, columnspan = 1)

# Generamos una etiqueta etiqueta_orto con el texto 'Orto:'.
etiqueta_orto = Tkinter.Label(segunda_ventana, text = 'Orto:',\
font = self.Verd11Roman)
# Asignamos una posición a la etiqueta en la segunda página.
etiqueta_orto.grid(row = 4, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)
# Evitamos la propagación de la etiqueta en la ventana.
etiqueta_orto.grid_propagate(0)

# En los sistemas operativos de la familia Windows el próximo orto
# de satélite seleccionado se mostrará en la segunda página de la
# Notebook. Crearemos los elementos necesarios a continuación.
# Creamos un objeto del tipo StringVar para mostrar el tiempo
# del orto del objeto.
self.text_var_ortobueno = Tkinter.StringVar()
# Como al iniciar el programa no habrá ningún objeto lo ponemos
# en blanco.
self.text_var_ortobueno.set('')

# Generamos una etiqueta para mostrar la información mediante la
# StringVar anterior.
mostrar_orto = Tkinter.Label(segunda_ventana,\
textvariable = self.text_var_ortobueno, font = self.Verd11Roman)
# Asignamos una posición a la etiqueta creada en la página.
mostrar_orto.grid(row = 4, column = 1, rowspan = 1, columnspan = 2)
# Cancelamos la propagación de la etiqueta.
```

```
            mostrar_orto.grid_propagate(0)

        # Creamos un botón para mostrar la posición del satélite elegido.
        # Este botón invocará a la función mostrar_una_vez_datos.
        self.boton_posicion = Tkinter.Button(ventana_controles,\
        text = 'Posición', font = self.Verd12Roman, height = 1, width = 11,\
        command = self.mostrar_unavez_datos)
        # Fijamos la posición del botón en el marco ventana_controles.
        self.boton_posicion.grid(row = 0, column = 1)
        # Este botón no estará disponible hasta que no seleccionemos un objeto
        # así que al crearlo lo deshabilitaremos.
        self.boton_posicion.config(state = Tkinter.DISABLED)

        # Creamos un botón para activar el seguimiento con el texto
        # 'Seguimiento'. Este botón esta enlazado con la función tack.
        self.boton_seguimiento = Tkinter.Button(ventana_controles,\
        text = 'Seguimiento', font = self.Verd12Roman, height = 1, width = 11,\
        command = self.tack)
        # Colocamos el botón boton_seguimiento en el marco ventana_controles.
        self.boton_seguimiento.grid(row = 1, column = 1)
        # Deshabilitamos boton_seguimiento hasta que no seleccionemos un objeto.
        self.boton_seguimiento.config(state = Tkinter.DISABLED)

        # Generamos un botón para parar el sistema con el texto 'Parada'.
        # Al pulsar el botón se ejecutará la función parada.
        boton_parada = Tkinter.Button(ventana_controles, text = 'Parada',\
        font = self.Verd12Roman, height = 1, width = 11, command = self.parada)
        # Asignamos una posición a boton_parada en el marco.
        boton_parada.grid(row = 2, column = 1)

        # Ya que la posición de los botones será distinta dependiendo del
        # sistema operativo tendremos que calcular el valor de la fila del
        # siguiente botón. Este cálculo lo realizaremos con la siguiente
        # sentencia.
        row_salida = row_nuevo + 1
        # Creamos el botón salir con el texto 'Salir' y enlazándolo a la
        # rutina salida.
        boton_salir = Tkinter.Button(ventana_controles, text = 'Salir',\
        font = self.Verd12Roman, height = 1, width = 11, command = self.salida)
        # Fijamos el botón boton_salir en el marco ventana_controles.
        boton_salir.grid(row = row_salida, column = 1)

    ##########################################################################
    ##########################################################################

    # Con la siguiente rutina crearemos una ventana donde estarán los controles
    # para actualizar la hora del sistema operativo.
    # La ventana está definidida aunque la función aún no ha sido implementada.
    def sincronizar_hora(self):

        # Creamos la ventana anidada ventana_hora.
        ventana_hora = Tkinter.Toplevel()
        # Definimos el tamaño de la ventana. Le damos un ancho de 250 píxeles
```

```
    # y una altura de 155. El margen interior será de 5 píxeles en sentido
    # horizontal y vertical.
    ventana_hora.geometry("250x155+5+5")
    # Fijamos el nombre de la ventana como 'Actualización de tiempo'.
    ventana_hora.title("Actualización de tiempo")

    # Generamos un botón para salir de la ventana. Este tiene el texto
    # 'Salir' y está enlazado a función destroy.
    boton_salir = Tkinter.Button(ventana_hora, text = 'Salir',\
    font = self.Verd12Roman, command = ventana_hora.destroy)
    # Asignamos una posición al boton boton_salir dentro de la ventana.
    boton_salir.grid(row = 0, column = 0, rowspan = 1, columnspan = 1)

    # Iniciamos la ventana ventana_hora.
    ventana_hora.mainloop()

##############################################################################
##############################################################################

# La función listado_posiciones creará una ventana anidada llamada
# ventana_posiciones para la gestión de la base de datos de localizaciones.
def listado_posiciones(self):

    # Creamos la ventana anidada ventana_posiciones.
    self.ventana_posiciones = Tkinter.Toplevel()
    # Fijamos el tamaño de la ventana en 450 píxeles de ancho y 180 píxeles
    # de alto. Asignamos un margen interior de 5 píxeles.
    self.ventana_posiciones.geometry("450x180+5+5")
    # Titulamos la ventana 'Listado de localizaciones'.
    self.ventana_posiciones.title("Listado de localizaciones")

    # Fijamos el tamaño mínimo de la primera columna en 170 píxeles.
    self.ventana_posiciones.columnconfigure(0, minsize = 170)
    # Fijamos el tamaño mínimo de la segunda columna en 170 píxeles.
    self.ventana_posiciones.columnconfigure(1, minsize = 170)
    # Fijamos el tamaño mínimo de la tercera columna en 40 píxeles.
    self.ventana_posiciones.columnconfigure(2, minsize = 40)

    ##########################################################################

    # Crearemos un marco que contendrá el listado de localizaciones del
    # sistema y los controles para añadir una nueva o modificar o eliminar
    # una existente.
    localizaciones = Tkinter.LabelFrame(self.ventana_posiciones,\
    text = 'Ubicaciones', font = self.Verd12Roman, width = 170,\
    height = 170)
    # Colocamos el marco anterior en la ventana ventana_posiciones.
    localizaciones.grid(row = 1, column = 0, rowspan = 3, columnspan = 1,\
    padx = 5, pady = 5)
    # Evitaremos el cambio de tamaño del marco.
    localizaciones.grid_propagate(0)

    # Fijamos el tamaño mínimo de la primera columna en 85 píxeles.
```

```
localizaciones.columnconfigure(0, minsize = 85)
# Fijamos el tamaño mínimo de la segunda columna en 85 píxeles.
localizaciones.columnconfigure(1, minsize = 85)

# Generamos un objeto del tipo ScrolledList para mostrar las
# localizaciones del sistema.
# Utilizaremos una sentencia if-elif para cada sistema operativo
# Sistemas operativos de la familia Macintosh.
if sys.platform == "darwin":
    # Creamos la lista de posiciones y la vinculamos a la función
    # mostrar_localizacion. Al pulsar sobre cada localización se
    # mostrarán los datos de la posición.
    self.lista_posiciones = scrolledlist.ScrolledList (localizaciones,\
    width=14, height=5, callback=self.mostrar_localizacion)
    # Le asignamos el tipo de letra Verd12Roman.
    self.lista_posiciones.listbox.configure(font = self.Verd12Roman)
    # Fijamos la posición de lista_posiciones.
    self.lista_posiciones.grid(row = 0, column = 0,
                               rowspan = 1, columnspan = 2)

# Sistemas operativos basados en el kernel de Linux.
elif sys.platform.startswith('linux'):
    # Creamos la lista de posiciones y la vinculamos a la función
    # mostrar_localizacion. Al pulsar sobre cada localización se
    # mostrarán los datos de la posición.
    self.lista_posiciones = scrolledlist.ScrolledList (localizaciones,\
    width=14, height=5, callback=self.mostrar_localizacion)
    # Le asignamos el tipo de letra Verd12Roman.
    self.lista_posiciones.listbox.configure(font = self.Verd12Roman)
    # Fijamos la posición de lista_posiciones.
    self.lista_posiciones.grid(row = 0, column = 0,
                               rowspan = 1, columnspan = 2)

# Sistemas operativos basados en Windows.
elif sys.platform == "win32":
    # Creamos la lista de posiciones y la vinculamos a la función
    # mostrar_localizacion. Al pulsar sobre cada localización se
    # mostrarán los datos de la posición.
    self.lista_posiciones = scrolledlist.ScrolledList (localizaciones,\
    width=16, height=5, callback=self.mostrar_localizacion)
    # Le asignamos el tipo de letra Verd12Roman.
    self.lista_posiciones.listbox.configure(font = self.Verd12Roman)
    # Fijamos la posición de lista_posiciones.
    self.lista_posiciones.grid(row = 0, column = 0,
                               rowspan = 1, columnspan = 2)

# Utilizando el módulo CSV abriremos el archivo localizaciones.list
# creando un objeto llamado listar_ciudades.
listar_ciudades = csv.reader(open("localizaciones.list", 'rb'))
# Leeremos todas las columnas del objeto anterior.
for row in listar_ciudades:
    # Guardaremos los valores del primer elemento de cada fila en el
    # objeto lista_posiciones.
```

```
        self.lista_posiciones.append(row[0])

    # Creamos los botones: boton_agregar, boton_modificar y boton_eliminar.
    # Con ellos podremos agregar una localización y modificar o eliminar
    # una existente.
    # La construcción de los botones será diferente dependiendo del sistema
    # operativo.
    # Sistemas operativos de la familia Macintosh.
    if sys.platform == "darwin":
        # Creamos un botón llamado boton_agregar con el texto 'Agregar
        # sitio' vinculado a la función agregar_ubicacion.
        boton_agregar = Tkinter.Button(localizaciones,\
        text = 'Agregar sitio', font = self.Verd12Roman, width = 16,\
        command = self.agregar_ubicacion)
        # Asignamos una posición a boton_agregar dentro del marco
        # localizaciones.
        boton_agregar.grid(row = 1, column = 0,
                           rowspan = 1, columnspan = 2)

        # Generamos un nuevo botón llamado boton_modificar con el texto
        # 'Modificar'.
        boton_modificar = Tkinter.Button(localizaciones,\
        text = 'Modificar', font = self.Verd12Roman, width = 6,\
        command = self.modificar_ubicacion)
        # Fijamos la posición de boton_modificar dentro del marco
        # localizaciones.
        boton_modificar.grid(row = 2, column = 0,
                             rowspan = 1, columnspan = 1)

        # Creamos el boton boton_eliminar con el texto 'Eliminar' que llama
        # a la función eliminar_ubicacion.
        boton_eliminar = Tkinter.Button(localizaciones,\
        text = 'Eliminar', font = self.Verd12Roman, width = 6,\
        command = self.eliminar_ubicacion)
        # Fijamos la posición de boton_eliminar en el marco localizaciones.
        boton_eliminar.grid(row = 2, column = 1,
                            rowspan = 1, columnspan = 1)

    # Distribuciones basadas en el kernel de Linux.
    elif sys.platform.startswith('linux'):
        # Creamos un botón llamado boton_agregar con el texto 'Agregar
        # sitio' vinculado a la función agregar_ubicacion.
        boton_agregar = Tkinter.Button(localizaciones,\
        text = 'Agregar sitio', font = self.Verd12Roman, width = 16,\
        command = self.agregar_ubicacion)
        # Asignamos una posición a boton_agregar dentro del marco
        # localizaciones.
        boton_agregar.grid(row = 1, column = 0,
                           rowspan = 1, columnspan = 2)

        # Generamos un nuevo botón llamado boton_modificar con el texto
        # 'Modificar'.
        boton_modificar = Tkinter.Button(localizaciones,\
```

```
        text = 'Modificar', font = self.Verd12Roman, width = 6,\
        command = self.modificar_ubicacion)
        # Fijamos la posición de boton_modificar dentro del marco
        # localizaciones.
        boton_modificar.grid(row = 2, column = 0,
                             rowspan = 1, columnspan = 1)

        # Creamos el boton boton_eliminar con el texto 'Eliminar' que llama
        # a la función eliminar_ubicacion.
        boton_eliminar = Tkinter.Button(localizaciones,\
        text = 'Eliminar', font = self.Verd12Roman, width = 6,\
        command = self.eliminar_ubicacion)
        # Fijamos la posición de boton_eliminar en el marco localizaciones.
        boton_eliminar.grid(row = 2, column = 1,
                            rowspan = 1, columnspan = 1)

    # Distribuciones de la serie Widnows.
    elif sys.platform == "win32":
        # Creamos un botón llamado boton_agregar con el texto 'Agregar
        # sitio' vinculado a la función agregar_ubicacion.
        boton_agregar = Tkinter.Button(localizaciones,\
        text = 'Agregar sitio', font = self.Verd12Roman, width = 18,\
        command = self.agregar_ubicacion)
        # Asignamos una posición a boton_agregar dentro del marco
        # localizaciones.
        boton_agregar.grid(row = 1, column = 0,
                           rowspan = 1, columnspan = 2)

        # Generamos un nuevo botón llamado boton_modificar con el texto
        # 'Modificar'.
        boton_modificar = Tkinter.Button(localizaciones,\
        text = 'Modificar', font = self.Verd12Roman, width = 8,\
        command = self.modificar_ubicacion)
        # Fijamos la posición de boton_modificar dentro del marco
        # localizaciones.
        boton_modificar.grid(row = 2, column = 0,
                             rowspan = 1, columnspan = 1)

        # Creamos el boton boton_eliminar con el texto 'Eliminar' que llama
        # a la función eliminar_ubicacion.
        boton_eliminar = Tkinter.Button(localizaciones,\
        text = 'Eliminar', font = self.Verd12Roman, width = 8,\
        command = self.eliminar_ubicacion)
        # Fijamos la posición de boton_eliminar en el marco localizaciones.
        boton_eliminar.grid(row = 2, column = 1,
                            rowspan = 1, columnspan = 1)

    ##########################################################################

    # Crearemos un objeto del tipo LabelFrame llamado caracteristicas donde
    # pondremos los widgets necesarios para mostrar la posición
    # seleccionada y modificar la posición que deseemos.
    self.caracteristicas = Tkinter.LabelFrame(self.ventana_posiciones,\
```

```
text = 'Características de la ubicación', font = self.Verd12Roman,\
width = 260, height = 118)
# Colocamos el marco creado en la ventana ventana_posiciones.
self.caracteristicas.grid(row = 1, column = 1, rowspan = 1,\
columnspan = 2, padx = 5, pady = 5, sticky = Tkinter.N)
# Evitamos el cambio de tamaño del marco en la ventana.
self.caracteristicas.grid_propagate(0)
# Fijamos un tamaño mínimo de 27 píxeles para la primera columna.
self.caracteristicas.columnconfigure(0, minsize = 27)
# Fijamos un tamaño mínimo de 27 píxeles para la segunda columna.
self.caracteristicas.columnconfigure(1, minsize = 27)
# Fijamos un tamaño mínimo de 27 píxeles para la tercera columna.
self.caracteristicas.columnconfigure(2, minsize = 27)
# Fijamos un tamaño mínimo de 27 píxeles para la cuarta columna.
self.caracteristicas.columnconfigure(3, minsize = 27)
# Fijamos un tamaño mínimo de 27 píxeles para la quinta columna.
self.caracteristicas.columnconfigure(4, minsize = 27)
# Fijamos un tamaño mínimo de 27 píxeles para la sexta columna.
self.caracteristicas.columnconfigure(5, minsize = 27)
# Fijamos un tamaño mínimo de 27 píxeles para la séptima columna.
self.caracteristicas.columnconfigure(6, minsize = 27)
# Fijamos un tamaño mínimo de 27 píxeles para la octava columna.
self.caracteristicas.columnconfigure(7, minsize = 27)
# Fijamos un tamaño mínimo de 27 píxeles para la novena columna.
self.caracteristicas.columnconfigure(8, minsize = 27)

# Creamos la etiqueta etiqueta_localizacion con el texto 'Localizacion'.
etiqueta_localizacion = Tkinter.Label(self.caracteristicas,\
text = 'Localización', font = self.Verd12Roman)
# Asignamos la posición de etiqueta_localizacion en el marco.
etiqueta_localizacion.grid(row = 0, column = 0,
                           rowspan = 1, columnspan = 3)

# Creamos una StringVar con el nombre text_var_localizacionmostrar para
# mostrar la localización.
self.text_var_localizacionmostrar = Tkinter.StringVar()
# Dejamos en blanco el texto de text_var_localizacionmostrar.
self.text_var_localizacionmostrar.set('')

# Generamos una etiqueta para mostrar la localización usando la
# StringVar anterior.
self.mostrar_localizacion = Tkinter.Label(self.caracteristicas,\
textvariable = self.text_var_localizacionmostrar,\
font = self.Verd12Roman)
# Asignamos una posición a la etiqueta en el marco caracteristicas.
self.mostrar_localizacion.grid(row = 1, column = 0,
                               rowspan = 1, columnspan = 3)
# Evitamos el cambio de tamaño de la etiqueta características.
self.mostrar_localizacion.grid_propagate(0)

# Creamos la etiqueta etiqueta_latitud con el texto 'Latitud'.
etiqueta_latitud = Tkinter.Label(self.caracteristicas,\
text = 'Latitud', font = self.Verd12Roman)
```

```
# Fijamos una posición para etiqueta_latitud en caracteristicas.
etiqueta_latitud.grid(row = 0, column = 3,
                      rowspan = 1, columnspan = 3)

# Generaremos una StringVar para mostrar la latitud.
self.text_var_latitudmostrar = Tkinter.StringVar()
# Ponemos el texto de la StringVar anterior en blanco.
self.text_var_latitudmostrar.set('')

# Creamos un objeto del tipo etiqueta para mostrar la latitud usando
# como texto la StringVar anterior.
self.mostrar_latitud = Tkinter.Label(self.caracteristicas,\
textvariable = self.text_var_latitudmostrar, font = self.Verd12Roman)
# Colocamos la etiqueta en el marco.
self.mostrar_latitud.grid(row = 1, column = 3,
                          rowspan = 1, columnspan = 3)
# Evitamos el cambio de tamaño de la etiqueta en el marco.
self.mostrar_latitud.grid_propagate(0)

# Creamos la etiqueta etiqueta_longitud con el texto 'Longitud'.
etiqueta_longitud = Tkinter.Label(self.caracteristicas,\
text = 'Longitud', font = self.Verd12Roman)
# Fijamos la posición de la etiqueta etiqueta_longitud en el marco.
etiqueta_longitud.grid(row = 0, column = 6,
                       rowspan = 1, columnspan = 3)

# Creamos una StringVar para mostrar la longitud.
self.text_var_longitudmostrar = Tkinter.StringVar()
# Ponemos en blanco el texto de la StringVar creada
self.text_var_longitudmostrar.set('')

# Generamos una etiqueta para mostrar la longitud utilizando la
# StringVar text_var_longitudmostrar
self.mostrar_longitud = Tkinter.Label(self.caracteristicas,\
textvariable = self.text_var_longitudmostrar, font = self.Verd12Roman)
# Asignamos una posición a la etiqueta en el marco.
self.mostrar_longitud.grid(row = 1, column = 6,
                           rowspan = 1, columnspan = 3)
# Evitamos el cambio de tamaño de la etiqueta.
self.mostrar_longitud.grid_propagate(0)

# Generamos una nueva etiqueta llamada etiqueta_altitud con el texto
# 'Altitud'.
etiqueta_altitud = Tkinter.Label(self.caracteristicas,\
text = 'Altitud', font = self.Verd12Roman)
# Colocamos la etiqueta en su posición en el marco.
etiqueta_altitud.grid(row = 2, column = 0, rowspan = 1, columnspan = 3)

# Creamos un objeto del tipo StringVar para mostrar la altitud.
self.text_var_altitudmostrar = Tkinter.StringVar()
# Ponemos en blanco el texto del objeto creado.
self.text_var_altitudmostrar.set('')
```

```
# Generamos una nueva etiqueta para mostrar la altitud utilizando el
# texto del StringVar anterior.
self.mostrar_altitud = Tkinter.Label(self.caracteristicas,\
textvariable = self.text_var_altitudmostrar, font = self.Verd12Roman)
# Fijamos la posición de la etiqueta en el marco.
self.mostrar_altitud.grid(row = 3, column = 0,
                          rowspan = 1, columnspan = 3)
# Evitamos la propagación de la etiqueta por el marco.
self.mostrar_altitud.grid_propagate(0)

# Creamos la etiqueta etiqueta_presion con el texto 'Presión'.
etiqueta_presion = Tkinter.Label(self.caracteristicas,\
text = 'Presión', font = self.Verd12Roman)
# Asignamos una posición la etiqueta en el marco características.
etiqueta_presion.grid(row = 2, column = 3, rowspan = 1, columnspan = 3)

# Creamos un objeto del tipo StringVar para mostrar la presión.
self.text_var_presionmostrar = Tkinter.StringVar()
# Ponemos en blanco el texto del StringVar.
self.text_var_presionmostrar.set('')

# Creamos una nueva etiqueta llamada mostrar_presion para mostrar
# la presión utilizando el StrinVar anterior.
self.mostrar_presion = Tkinter.Label(self.caracteristicas,\
textvariable = self.text_var_presionmostrar, font = self.Verd12Roman)
# Asignamos una posición al objeto en el marco caracteristicas.
self.mostrar_presion.grid(row = 3, column = 3,
                          rowspan = 1, columnspan = 3)
# Evitamos la propagación de la etiqueta en el marco.
self.mostrar_presion.grid_propagate(0)

# Generamos una nueva etiqueta con el texto 'Horizonte'.
etiqueta_horizonte = Tkinter.Label(self.caracteristicas,\
text = 'Horizonte', font = self.Verd12Roman)
# Fijamos la posición de la etiqueta en el marco.
etiqueta_horizonte.grid(row = 2, column = 6,
                        rowspan = 1, columnspan = 3)

# Creamos un StringVar para mostrar el horizonte.
self.text_var_horizontemostrar = Tkinter.StringVar()
# Ponemos el texto del StringVar en blanco.
self.text_var_horizontemostrar.set('')

# Creamos una etiqueta llamada mostrar_horizonte para mostrar el
# horizonte usando como texto el StringVar anterior.
self.mostrar_horizonte = Tkinter.Label(self.caracteristicas,\
textvariable = self.text_var_horizontemostrar, font = self.Verd12Roman)
# Asignamos una posición a la etiqueta en el marco.
self.mostrar_horizonte.grid(row = 3, column = 6,
                            rowspan = 1, columnspan = 3)
# Cancelamos el cambio de tamaño de la etiqueta que hemos creado.
self.mostrar_horizonte.grid_propagate(0)
```

```
        ##########################################################################

        # Necesitamos una etiqueta para mostrar los avisos del programa cuando
        # modifique nuestra base de datos de localizaciones.
        # Generamos un nuevo objeto del tipo StringVar para estos avisos.
        self.text_var_avisobueno = Tkinter.StringVar()
        # Asignamos un texto inicial al objeto 'Avisos del sistema'.
        self.text_var_avisobueno.set('Avisos del sistema.')

        # Creamos una etiqueta para mostrar los avisos usando como texto
        # el StringVar text_var_avisobueno.
        self.mostrar_avisos = Tkinter.Label(self.ventana_posiciones,\
        textvariable = self.text_var_avisobueno, font = self.Verd12Roman)
        # Colocamos la etiqueta en el marco.
        self.mostrar_avisos.grid(row = 2, column = 1,
                                 rowspan = 1, columnspan = 2)

        # Necesitamos un botón para aceptar las modificaciones realizadas.
        # Para ahorrar espacio en la pantalla usaremos siempre el mismo botón.
        # Evitaremos confusiones cambiando el texto del botón dependiendo de
        # la acción que querramos realizar.
        # Tendremos primero que crear un objeto del tipo StringVar.
        self.text_var_aceptarconfirmacion = Tkinter.StringVar('')
        # A continuación crearemos el botón vinculándolo funcion_variable.
        self.boton_aceptar = Tkinter.Button(self.ventana_posiciones,\
        textvariable = self.text_var_aceptarconfirmacion,\
        font = self.Verd12Roman, width = 18, command = self.funcion_variable)
        # Colocamos el botón en la pantalla.
        self.boton_aceptar.grid(row = 3, column = 1,
                                rowspan = 1, columnspan = 1)
        # Ya que el botón solo será necesario una vez hallamos seleccionado una
        # acción tendremos que deshabilitar el botón hasta aquel entonces.
        self.boton_aceptar.config(state = Tkinter.DISABLED)

        # Creamos un botón para salir de la ventana. Este tendrá el texto
        # 'Salir' y estará vinculado a salir_ventana_localizacion, función
        # encargada de cerrar la ventana.
        boton_salir = Tkinter.Button(self.ventana_posiciones, text = 'Salir',\
        font = self.Verd12Roman, command = self.salir_ventana_localizacion,\
        width = 8)
        # Fijamos la posición del botón en el marco.
        boton_salir.grid(row = 3, column = 2, rowspan = 1, columnspan = 1)

        # Iniciamos la ventana ventana_posiciones mediante la siguiente
        # sentencia.
        self.ventana_posiciones.mainloop()

    ##############################################################################
    # A continuación crearemos las funciones necesarias para agregar, modificar
    # y eliminar ubicaciones.
    # La primera rutina que definiremos será la necesaria para mostrar una
    # localización por pantalla.
    # La función mostrar_localizacion recogerá como argumento el valor de
```

```
# indice_localizacion.
def mostrar_localizacion(self, indice_localizacion):
    # Abrimos el archivo localizaciones.list utilizando el módulo CSV.
    listar_observador = csv.reader(open("localizaciones.list", 'rb'))

    self.indice_localizacion_extraccion = indice_localizacion

    # Volvemos a mostrar el aviso del sistema estandar.
    self.text_var_avisobueno.set('Avisos del sistema')

    # Creamos un bucle for que recorrerá todas las filas del fichero hasta
    # llegar a la correspondiente a la localización escogida.
    # Ya que la primera fila es la número 1 y el primer indice es el número
    # 0 tendremos que sumar uno al valor del índice para que ambos registros
    # se correspondan.
    for i in range(indice_localizacion + 1):
        # Cada vez que se repita el bucle el valor de observador se
        # corresponderá con la siguiente línea del fichero.
        observador = listar_observador.next()
        # Los valores en observador son almacenados con la forma de una
        # lista así que iremos asignando cada elemento de esta a su
        # etiqueta correspondiente.
        # El nombre de la localización será el primer elemento. Asignaremos
        # el valor de este al StringVar text_var_localizacionmostrar.
        self.text_var_localizacionmostrar.set(observador[0])
        # La latitud estará en el segundo, tercer y cuarto elemento.
        # Formatearemos el valor antes de pasarselo al StringVar
        # text_var_latitudmostrar.
        self.text_var_latitudmostrar.set("%s:%s:%s" % (observador[1],\
        observador[2], observador[3]))
        # La longitud se encontrará en los valores quinto, sexto y séptimo.
        # Estos se asignarán, una vez formateados, al StringVar
        # text_var_longitudmostrar.
        self.text_var_longitudmostrar.set("%s:%s:%s" %(observador[4],\
        observador[5], observador[6]))
        # La altitud será el octavo elemento de la lista, en este caso no
        # formatearemos el valor antes de pasarselo al StringVar
        # text_var_altitudmostrar.
        self.text_var_altitudmostrar.set("%s" %(observador[7]))
        # La presión se encontrará en el noveno elemento, el valor de este
        # se colocará en el StringVar text_var_presionmostrar.
        self.text_var_presionmostrar.set("%s" %(observador[8]))
        # El horizonte estará en la última posición, la décima, del
        # conjunto. Esta irá al StringVar text_var_horizontemostrar.
        self.text_var_horizontemostrar.set("%s" %(observador[9]))

# La siguiente función ocultará de la pantalla las etiquetas encargadas de
# mostrar las valores de las localizaciones y colocará, en su lugar, los
# campos de entrada necesarios para introducir los nuevos valores.
def modificar_widgets(self):

    # Ocultamos la etiqueta mostrar_localizacion.
    self.mostrar_localizacion.grid_remove()
```

```
# Ocultamos la etiqueta mostrar_latitud.
self.mostrar_latitud.grid_remove()
# Ocultamos la etiqueta mostrar_longitud.
self.mostrar_longitud.grid_remove()
# Ocultamos la etiqueta mostrar_altitud.
self.mostrar_altitud.grid_remove()
# Ocultamos la etiqueta mostrar_presion.
self.mostrar_presion.grid_remove()
# Ocultamos la etiqueta mostrar_horizonte.
self.mostrar_horizonte.grid_remove()

# Creamos un objeto de la clase Entry para introducir el nombre de
# la localización del observador.
self.introducir_localizacion = Tkinter.Entry(self.caracteristicas,\
width = 8, font = self.Verd12Roman)
# Fijamos la posición del objeto en el marco características.
self.introducir_localizacion.grid(row = 1, column = 0,
                                  rowspan = 1, columnspan = 3)

# En este punto tendremos que realizar una pequeña aclaración.
# En este programa la longitud y la latitud se muestran como un único
# valor formateados de la forma "grados:minutos:segundos".
# Cuando introduzcamos nuevos valores de longitud y latitud no tendremos
# que seguir este esquema ya que el script creará campos independientes
# para los grados, minutos y segundos.
# Así, a continuación, generamos tres objetos de la clase Entry para
# introducir los grados, minutos y segundos de la
# latitud del observador.
# Crearemos el objeto introducir_latitud1 para los grados.
self.introducir_latitud1 = Tkinter.Entry(self.caracteristicas,\
width = 2, font = self.Verd12Roman)
# Colocaremos el objeto en el marco.
self.introducir_latitud1.grid(row = 1, column = 3,
                              rowspan = 1, columnspan = 1)
# El objeto introducir_latitud2 se encargará de los minutos.
self.introducir_latitud2 = Tkinter.Entry(self.caracteristicas,\
width = 2, font = self.Verd12Roman)
# Le asignaremos una posición al lado del anterior objeto.
self.introducir_latitud2.grid(row = 1, column = 4,
                              rowspan = 1, columnspan = 1)
# Por último lugar el objeto introducir_latitud3 se encargará de los
# segundos de la latitud.
self.introducir_latitud3 = Tkinter.Entry(self.caracteristicas,\
width = 3, font = self.Verd12Roman)
# Lo colocaremos en último lugar.
self.introducir_latitud3.grid(row = 1, column = 5,
                              rowspan = 1, columnspan = 1)

# Con la longitud realizaremos la misma operación. En primer lugar
# generaremos el objeto destinado a los grados de la longitud.
self.introducir_longitud1 = Tkinter.Entry(self.caracteristicas,\
width = 2, font = self.Verd12Roman)
# Asignaremos una posición al objeto a continuación de la latitud.
```

```
        self.introducir_longitud1.grid(row = 1, column = 6,
                                       rowspan = 1, columnspan = 1)
        # De los minutos se encargará introducir_longitud2.
        self.introducir_longitud2 = Tkinter.Entry(self.caracteristicas,\
        width = 2, font = self.Verd12Roman)
        # Fijaremos la posición del objeto a continuación de los grados.
        self.introducir_longitud2.grid(row = 1, column = 7,
                                       rowspan = 1, columnspan = 1)
        # Para introducir los segundos recurriremos al objeto
        # introducir_longitud3.
        self.introducir_longitud3 = Tkinter.Entry(self.caracteristicas,\
        width = 3, font = self.Verd12Roman)
        # Este objeto irá al final de la serie de la longitud.
        self.introducir_longitud3.grid(row = 1, column = 8,
                                       rowspan = 1, columnspan = 1)

        # Creamos un objeto de la clase Entry para la altitud del observador.
        self.introducir_altitud = Tkinter.Entry(self.caracteristicas,\
        width = 8, font = self.Verd12Roman)
        # Asignamos una posición al objeto creado en el marco.
        self.introducir_altitud.grid(row = 3, column = 0,
                                     rowspan = 1, columnspan = 3)

        # Generamos un objeto llamado introducir_presion para guardar la
        # presión media de la localización del observador.
        self.introducir_presion = Tkinter.Entry(self.caracteristicas,\
        width = 8, font = self.Verd12Roman)
        # Fijamos la posición del objeto introducir_presion en el marco.
        self.introducir_presion.grid(row = 3, column = 3,
                                     rowspan = 1, columnspan = 3)

        # Por último creamos el objeto necesario para asignar un valor al
        # horizonte del observador.
        self.introducir_horizonte = Tkinter.Entry(self.caracteristicas,\
        width = 8, font = self.Verd12Roman)
        # Colocaremos el objeto en el marco características.
        self.introducir_horizonte.grid(row = 3, column = 6,
                                       rowspan = 1, columnspan = 3)

    # Cuando pulsemos el botón con el texto 'Modificar' llamaremos a la
    # siguiente rutina.
    # Esta se encargará de preparar nuestro programa para la modificación de
    # la ubicación seleccionada.
    def modificar_ubicacion(self):
        # Activamos el boton para aceptar los cambios.
        self.boton_aceptar.config(state = Tkinter.ACTIVE)
        # Cambiamos la etiqueta del botón.
        self.text_var_aceptarconfirmacion.set('Aceptar cambio')
        # Llamamos a la función para crear los campos de entrada de texto.
        self.modificar_widgets()

    # Al presionar el botón con el texto 'Agregar ubicación' el programa
    # ejecutará la siguiente rutina.
```

```
# El script ejecutará las rutinas necesarias para agregar una nueva
# localización a la base de datos.
def agregar_ubicacion(self):
    # Activamos el boton para aceptar la nueva ubicación.
    self.boton_aceptar.config(state = Tkinter.ACTIVE)
    # Cambiamos la etiqueta del boton.
    self.text_var_aceptarconfirmacion.set('Aceptar ubicación')
    # Llamamos a la función para crear los campos de entrada de texto.
    self.modificar_widgets()

# Cuando presionamos el boton boton_cambiar se ejecuta la función siguiente.
# Esta nos sirve para elegir que rutina tendremos que ejecutar a
# continuacón, para eso se valdrá del texto del botón.
# TO-DO. Aunque este método es válido si utilizaramos un flag para esto
# quizás el programa sería más ágil.
def funcion_variable(self):
    # Obtenemos el texto del botón mediante el método get().
    variable = self.text_var_aceptarconfirmacion.get()
    # Si el texto es 'Aceptar cambio' ejecutaremos aceptar_modificacion.
    if variable == 'Aceptar cambio':
        self.aceptar_modificacion()
    # Si el texto es 'Aceptar ubicacion' ejecutaremos aceptar_ubicacion.
    elif variable == 'Aceptar ubicacion':
        self.aceptar_ubicacion()
    # Si el botón no tiene ningún texto no ejecutaremos nada.
    else:
        pass

# La siguiente función se encargará de almacenar una nueva ubicación en la
# base de datos.
def aceptar_ubicacion(self):
    # Obtenemos los datos de la localizacion de los objetos de la clase
    # Entry asignamos cada dato a una nueva variable.
    # Guardamos la localización en la variable localizacion.
    localizacion = self.introducir_localizacion.get()
    # A continuación guardamos los grados, minutos y segundos de la latidud
    # en las variables latitud1, latitud2 y latitud3.
    latitud1 = self.introducir_latitud1.get()
    latitud2 = self.introducir_latitud2.get()
    latitud3 = self.introducir_latitud3.get()
    # Los grados, minutos y segundos de la longitud irán a las variables
    # longitud1, longitud2 y longitud3.
    longitud1 = self.introducir_longitud1.get()
    longitud2 = self.introducir_longitud2.get()
    longitud3 = self.introducir_longitud3.get()
    # La altitud de la localización irá a la variable altitud.
    altitud = self.introducir_altitud.get()
    # Asignaremos el valor de la presión media a presion.
    presion = self.introducir_presion.get()
    # Por último lugar el valor del horizonte irá a la variable horizonte.
    horizonte = self.introducir_horizonte.get()

    # Agregamos los datos al fichero de localizaciones.
```

```
        # Para ellos recurriremos de nuevo al módulo CSV. Abriremos el archivo
        # 'localizaciones.list' con el método 'ab'. Es decir, agregaremos una
        # nueva fila al final del fichero con nuestros valores.
        agregar_datos = csv.writer(open("localizaciones.list", 'ab'))
        # Escribiremos la nueva fila en el fichero.
        agregar_datos.writerow([localizacion, latitud1, latitud2, latitud3,\
        longitud1, longitud2, longitud3, altitud, presion, horizonte])

        # TO-DO. La fila se almacena sin ningún orden. Debería almacenarse por
        # orden alfabético.

        # Mostramos un aviso en la ventana.
        self.text_var_avisobueno.set('Localizacion agregada')

        # Eliminamos los campos de introducción de texto.
        # El correspondiente al nombre de la ubicación.
        self.introducir_localizacion.grid_remove()
        # Los correspondientes a la latitud del observador.
        self.introducir_latitud1.grid_remove()
        self.introducir_latitud2.grid_remove()
        self.introducir_latitud3.grid_remove()
        # Los asignados a la longitud de la localización.
        self.introducir_longitud1.grid_remove()
        self.introducir_longitud2.grid_remove()
        self.introducir_longitud3.grid_remove()
        # El correspondiente a la altura de la localización.
        self.introducir_altitud.grid_remove()
        # El correspondiente a la presión media de la ubicación.
        self.introducir_presion.grid_remove()
        # El asignado al horizonte del observador.
        self.introducir_horizonte.grid_remove()

        # Recupero la interfaz anterior volviendo a mostrar las etiquetas
        # que muestran los valores de las ubicaciones.
        # Localización del observador.
        self.mostrar_localizacion.grid()
        # Latitud del observador.
        self.mostrar_latitud.grid()
        # Longitud del observador.
        self.mostrar_longitud.grid()
        # Altitud del observador.
        self.mostrar_altitud.grid()
        # Presión media en la ubicación del observador.
        self.mostrar_presion.grid()
        # Altura del horizonte desde la localización.
        self.mostrar_horizonte.grid()

    # Modificar una ubicación será un proceso algo más complicado y es que
    # tendremos que eliminar las antiguas ubicaciones para substituirlas por
    # unas nuevas.
    # La siguiente rutina se encargará de ello.
    # TO-DO. Utilizar un sistema de bases de datos más eficiente. Este puede
    # dar problemas en el futuro.
```

```
def aceptar_modificacion(self):

    # Utilizando el módulo CSV leeremos todos los valores del fichero
    # localizaciones.list almacenando los valores en listar_observador.
    listar_observador = csv.reader(open("localizaciones.list", 'rb'))

    # Creamos diferentes listas para almacenar los valores del fichero.
    # Nombres de las localizaciones.
    localizacion = []
    # Grados de las diferentes latitudes.
    latitud1 = []
    # Minutos de las latitudes guardadas.
    latitud2 = []
    # Segundos de las diferentes latitudes.
    latitud3 = []
    # Grados de las longitudes almacenadas en la base de datos.
    longitud1 = []
    # Minutos de las diferentes longitudes.
    longitud2 = []
    # Segundos de las longitude guardadas.
    longitud3 = []
    # Diferentes altitudes almacenadas.
    altitud = []
    # Presiones medias guardadas en la base de datos.
    presion = []
    # Altura de los horizontes de los observadores.
    horizonte = []

    # Leeremos el fichero listar_observador y guardaremos cada uno de los
    # registros de este en las listas creadas anteriormente.
    for row in listar_observador:
        localizacion.append(row[0])
        latitud1.append(row[1])
        latitud2.append(row[2])
        latitud3.append(row[3])
        longitud1.append(row[4])
        longitud2.append(row[5])
        longitud3.append(row[6])
        altitud.append(row[7])
        presion.append(row[8])
        horizonte.append(row[9])

    # A continuación eliminaremos el registro deseado. Para ello
    # eliminaremos de cada lista el elemento correspondiente al registro.
    # La localización del elemento en cada lista nos la dará el indice
    # indice_localizacion_extraccion.
    # Eliminamos el nombre de la localización.
    localizacion.pop(self.indice_localizacion_extraccion)
    # Borramos los valores de la latitud.
    latitud1.pop(self.indice_localizacion_extraccion)
    latitud2.pop(self.indice_localizacion_extraccion)
    latitud3.pop(self.indice_localizacion_extraccion)
    # Eliminamos los valores de la longitud.
```

```
longitud1.pop(self.indice_localizacion_extraccion)
longitud2.pop(self.indice_localizacion_extraccion)
longitud3.pop(self.indice_localizacion_extraccion)
# Eliminamos la altitud de la localización.
altitud.pop(self.indice_localizacion_extraccion)
# Borramos la presión media en la ubicacion.
presion.pop(self.indice_localizacion_extraccion)
# Borramos el valor del horizonte del observador.
horizonte.pop(self.indice_localizacion_extraccion)


# A continuación tendremos que obtener los nuevos valores de los campos
# de entrada de texto.
# Nombre de la ubicación.
elemento_localizacion = self.introducir_localizacion.get()
# Grados, minutos y segundos de la latitud.
elemento_latitud1 = self.introducir_latitud1.get()
elemento_latitud2 = self.introducir_latitud2.get()
elemento_latitud3 = self.introducir_latitud3.get()
# Grado, minutos y segundos de la longitud.
elemento_longitud1 = self.introducir_longitud1.get()
elemento_longitud2 = self.introducir_longitud2.get()
elemento_longitud3 = self.introducir_longitud3.get()
# Altura en metros en la ubicación.
elemento_altitud = self.introducir_altitud.get()
# Presión media en la localización del observador.
elemento_presion = self.introducir_presion.get()
# Horizonte medio en la ubicación del observador.
elemento_horizonte = self.introducir_horizonte.get()


# Agregaramos los datos obtenidos introduciendo en cada lista su valor
# correspondiente en el indice indice_localizacion_extraccion.
# Empezaremos por el nombre de la localización.
localizacion.insert(self.indice_localizacion_extraccion,
                    elemento_localizacion)
# Grados, minutos y segundos de la latitud.
latitud1.insert(self.indice_localizacion_extraccion,
                elemento_latitud1)
latitud2.insert(self.indice_localizacion_extraccion,
                elemento_latitud2)
latitud3.insert(self.indice_localizacion_extraccion,
                elemento_latitud3)
# Grados, minuto y segundos de la longitud.
longitud1.insert(self.indice_localizacion_extraccion,
                 elemento_longitud1)
longitud2.insert(self.indice_localizacion_extraccion,
                 elemento_longitud2)
longitud3.insert(self.indice_localizacion_extraccion,
                 elemento_longitud3)
# Altitud de la localización.
altitud.insert(self.indice_localizacion_extraccion,
               elemento_altitud)
```

```
# Presión atmosférica media en la ubicación del observador.
presion.insert(self.indice_localizacion_extraccion,
                elemento_presion)
# Altura del horizonte.
horizonte.insert(self.indice_localizacion_extraccion,
                  elemento_horizonte)

# Creamos una lista llamada observador para guardar las listas.
observador = []

# Mediante un bucle for guardamos los valores de las listas en nuestra
# lista observador.
for i in range(len(localizacion)):
    observador.append([localizacion[i], latitud1[i], latitud2[i],\
    latitud3[i], longitud1[i], longitud2[i], longitud3[i], altitud[i],\
    presion[i], horizonte[i]])


# Agregamos la lista a la base de datos utilizando el módulo CSV.
agregar_datos = csv.writer(open("localizaciones.list", 'wb'))
agregar_datos.writerows(observador)

# Mostramos un aviso por la GUI
self.text_var_avisobueno.set('Localizacion modificada')

# Eliminamos los campos de introducción de texto.
# El correspondiente al nombre de la ubicación.
self.introducir_localizacion.grid_remove()
# Los correspondientes a la latitud del observador.
self.introducir_latitud1.grid_remove()
self.introducir_latitud2.grid_remove()
self.introducir_latitud3.grid_remove()
# Los asignados a la longitud de la localización.
self.introducir_longitud1.grid_remove()
self.introducir_longitud2.grid_remove()
self.introducir_longitud3.grid_remove()
# El correspondiente a la altura de la localización.
self.introducir_altitud.grid_remove()
# El correspondiente a la presión media de la ubicación.
self.introducir_presion.grid_remove()
# El asignado al horizonte del observador.
self.introducir_horizonte.grid_remove()

# Recupero la interfaz anterior volviendo a mostrar las etiquetas
# que muestran los valores de las ubicaciones.
# Localización del observador.
self.mostrar_localizacion.grid()
# Latitud del observador.
self.mostrar_latitud.grid()
# Longitud del observador.
self.mostrar_longitud.grid()
# Altitud del observador.
self.mostrar_altitud.grid()
```

```
        # Presión media en la ubicación del observador.
        self.mostrar_presion.grid()
        # Altura del horizonte desde la localización.
        self.mostrar_horizonte.grid()

        # Volvemos a deshabilitar el botón boton_aceptar para evitar que se
        # pueda ejecutar de nuevo esta rutina.
        self.boton_aceptar.config(state = Tkinter.DISABLED)

        # TO-DO. Cuando termino la rutina no actualiza el listado, solucionarlo.

    # La siguiente función se encargara de eliminar la ubicación seleccionada.
    # Recurrirá también, como en los casos anteriores, al uso del módulo CSV
    # y a la utilización de las listas.
    def eliminar_ubicacion(self):
        # Leemos el contenido de la base de datos y lo almacenamos en la
        # variable listar_observador.
        listar_observador = csv.reader(open("localizaciones.list", 'rb'))

        # Creamos diferentes listas para almacenar los valores del fichero.
        # Nombres de las localizaciones.
        localizacion = []
        # Grados de las diferentes latitudes.
        latitud1 = []
        # Minutos de las latitudes guardadas.
        latitud2 = []
        # Segundos de las diferentes latitudes.
        latitud3 = []
        # Grados de las longitudes almacenadas en la base de datos.
        longitud1 = []
        # Minutos de las diferentes longitudes.
        longitud2 = []
        # Segundos de las longitudes guardadas.
        longitud3 = []
        # Diferentes altitudes almacenadas.
        altitud = []
        # Presiones medias guardadas en la base de datos.
        presion = []
        # Altura de los horizontes de los observadores.
        horizonte = []

        # Leeremos el fichero listar_observador y guardaremos cada uno de los
        # registros de este en las listas creadas anteriormente.
        for row in listar_observador:
            localizacion.append(row[0])
            latitud1.append(row[1])
            latitud2.append(row[2])
            latitud3.append(row[3])
            longitud1.append(row[4])
            longitud2.append(row[5])
            longitud3.append(row[6])
            altitud.append(row[7])
            presion.append(row[8])
```

```
            horizonte.append(row[9])

        # A continuación eliminaremos el registro deseado. Para ello
        # eliminaremos de cada lista el elemento correspondiente al registro.
        # La localización del elemento en cada lista nos la dará el indice
        # indice_localizacion_extraccion.
        # Eliminamos el nombre de la localización.
        localizacion.pop(self.indice_localizacion_extraccion)
        # Borramos los valores de la latitud.
        latitud1.pop(self.indice_localizacion_extraccion)
        latitud2.pop(self.indice_localizacion_extraccion)
        latitud3.pop(self.indice_localizacion_extraccion)
        # Eliminamos los valores de la longitud.
        longitud1.pop(self.indice_localizacion_extraccion)
        longitud2.pop(self.indice_localizacion_extraccion)
        longitud3.pop(self.indice_localizacion_extraccion)
        # Eliminamos la altitud de la localización.
        altitud.pop(self.indice_localizacion_extraccion)
        # Borramos la presión media en la ubicacion.
        presion.pop(self.indice_localizacion_extraccion)
        # Borramos el valor del horizonte del observador.
        horizonte.pop(self.indice_localizacion_extraccion)

        # Creo una nueva lista para almacenar las listas anteriores.
        observador = []

        # Mediante un bucle for guardamos los valores de las listas en nuestra
        # lista observador.
        for i in range(len(localizacion)):
            observador.append([localizacion[i], latitud1[i], latitud2[i],\
            latitud3[i], longitud1[i], longitud2[i], longitud3[i], altitud[i],\
            presion[i], horizonte[i]])

        # Agregamos los nuevos datos al fichero de localizaciones
        agregar_datos = csv.writer(open("localizaciones.list", 'wb'))
        agregar_datos.writerows(observador)

        # Mostramos un aviso por la ventana.
        self.text_var_avisobueno.set('Localizacion eliminada')

    # Esta función nos permitirá salir del programa.
    def salir_ventana_localizacion(self):
        # El método destroy() eliminará la ventana y todos los procesos
        # asociados a esta.
        self.ventana_posiciones.destroy()

    ###########################################################################
    ###########################################################################

    # Rutina para acceder a la ventana de configuracion
    def configurar_sistema(self):
        # Creamos una ventana anidada llamada ventana_configuracion.
        ventana_configuracion = Tkinter.Toplevel()
```

```
# Definimos el tamaño de la ventana. Le daremos 600 píxeles de ancho por
# 373 píxeles de alto. El margen interior será de 5 píxeles.
ventana_configuracion.geometry("600x373+5+5")
# Nombramos a la ventana 'Configuración del sistema'.
ventana_configuracion.title("Configuracion del sistema")

# Creamos la notebook para almacenar las distintas páginas con las
# dversas configuraciones.
configuracion_notebook = ttk.Notebook(ventana_configuracion,
                                      width = 550, height = 323)
# Colocamos la notebook en la ventana.
configuracion_notebook.grid(row = 0, column = 0, rowspan = 1,
                            columnspan = 1, sticky = Tkinter.S)
# Evitamos el cambio de tamaño de la notebook.
configuracion_notebook.grid_propagate(0)

########################################################################

# En la primera página de la notebook tendremos los controles necesarios
# para gestionar la base de datos de elementos orbitales de sistema.
# Creamos la primera página asignándole el nombres tles_pagina.
tles_pagina = ttk.Frame(configuracion_notebook)
# Nombramos a tles_pagina con el texto 'TLEs'.
configuracion_notebook.add(tles_pagina, text='TLEs')
# Asignamos a la primera columna un tamaño mínimo de 290 píxeles.
tles_pagina.columnconfigure(0, minsize = 290)
# Asignamos a la segunda columna un tamaño mínmo de 45 píxeles.
tles_pagina.columnconfigure(1, minsize = 45)

# Generamos un nuevo objeto del tipo LabelFrame en la primera página
# del notebook
ventana_tles = Tkinter.LabelFrame(tles_pagina,\
text = 'TLEs disponibles', width = 280, height = 290,\
font = self.Verd12Italic)
# Asignamos una posición al objeto anteriromente creado.
ventana_tles.grid(row = 0, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)
# Evitamos el cambio de tamaño del objeto.
ventana_tles.grid_propagate(0)
# Configuramos el ancho minimo de la primera columna del objeto.
ventana_tles.columnconfigure(0, minsize = 61)
# Configuramos el ancho mínimo de la segunda columna del objeto.
ventana_tles.columnconfigure(1, minsize = 61)
# Configuramos el ancho mínimo de la tercera columna del objeto.
ventana_tles.columnconfigure(2, minsize = 61)

# A continuación generaremos una serie de objetos del tipo Checkbutton
# que nos permitirán hacer la selección de las familias.
# Cada selección corresponderá a una familia de satélites distinta.
# Realizaremos una nueva sentencia del tipo if-elif para seleccionar
# entre los comportamientos necesarios para cada sistema operativo.
# Sistemas operativos de la familia Macintosh
if sys.platform == "darwin":
```

```
# A continuación crearemos cada Checkbutton. El proceso de creación
# de cada una de ellos será muy similar. Primero generamos un
# nuevo objeto del tipo StringVar. Ya que la cadena de texto que
# devolverán los Checkbutton será distinta dependiendo de si están
# seleccionados o no necesitaremos un objeto de texto que cambie
# de valor dinamicamente.
# Hecho esto generamos el objeto de tipo Checkbutton asignando los
# valores de activado y desactivado correspondientes y colocando
# el objeto en su lugar.
# También vincularemos nuestro objeto a la función mostrar_satelites
# lo cual nos permitirá mostrar los satélites disponibles en cada
# familia en una lista del tipo ScrolledList.

# Creación del objeto de tipo StringVar variable_weather.
self.variable_weather = Tkinter.StringVar()
# Creación del objeto de tipo Checkbutton weather.
# Definimos como valor para cuando este activado 'weather.txt' y '0'
# para cuando no esté activado.
# Al pulsar el objeto pasaremos la cadena de caracteres
# 'weather.txt' a la función mostrar_satelites.
self.weather = Tkinter.Checkbutton(ventana_tles, text = 'Weather',\
font = self.Verd11Roman, variable = self.variable_weather,\
onvalue = 'weather.txt', offvalue = 0,\
command = lambda flag = 'weather.txt': self.mostrar_satelites(flag))
# Asignamos una posición al objeto creado.
self.weather.grid(row = 0, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos el objeto variable_noaa del tipo StringVar.
self.variable_noaa = Tkinter.StringVar()
# Creamos un Checkbutton llamado noaa. El valor que nos devolverá
# al activarse será 'noaa.txt' devolviéndonos '0' al desactivarse.
# Invocaremos a la función mostrar_satelites con el texto
# 'noaa.txt' al pulsar sobre el objeto.
self.noaa = Tkinter.Checkbutton(ventana_tles, text = 'Noaa',\
font = self.Verd11Roman, variable = self.variable_noaa,\
onvalue = 'noaa.txt', offvalue = 0,\
command = lambda flag = 'noaa.txt': self.mostrar_satelites(flag))
# Asignamos una posición al objeto en la ventana.
self.noaa.grid(row = 0, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un objeto variable_goes del tipo StringVar.
self.variable_goes = Tkinter.StringVar()
# Crearemos un objeto llamado goes del tipo Checkbutton.
# Este objeto nos devolverá la cadena de caracteres 'goes.txt'
# al activar el objeto y '0' al desactivarlo.
# Al pulsarlo, en cualquiera de las ocasiones, llamaremos a la
# función mostrar_satelites con el texto 'goes.txt'.
self.goes = Tkinter.Checkbutton(ventana_tles, text = 'Goes',\
font = self.Verd11Roman, variable = self.variable_goes,\
onvalue = 'goes.txt', offvalue = 0,\
command = lambda flag = 'goes.txt': self.mostrar_satelites(flag))
```

```
# Fijamos una posición para el objeto en el LabelFrame ventana_tles.
self.goes.grid(row = 0, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un objeto del tipo StringVar llamado variable_resource.
self.variable_resource = Tkinter.StringVar()
# Generamos un nuevo objeto del tipo Checkbutton. El texto que nos
# devolverá al activarlo será 'resource.txt' siendo '0' al
# desactivarlo.
# Al hacer click en el objeto pasaremos el texto 'resource.txt' a
# la función mostrar_satelites.
self.resource = Tkinter.Checkbutton(ventana_tles,\
text = 'Resource', font = self.Verd11Roman,\
variable = self.variable_resource, onvalue = 'resource.txt',\
offvalue = 0,\
command = lambda flag = 'resource.txt':\
self.mostrar_satelites(flag))
# Asignamos una posición al objeto creado en el objeto LabelFrame.
self.resource.grid(row = 1, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un nuevo objeto del tipo StringVar llamado
# variable_sarsat.
self.variable_sarsat = Tkinter.StringVar()
# Creamos un objeto llamado sarsat del tipo Checkbutton.
# Este objeto nos devolverá el texto 'sarsat.txt' cuando se
# encuentre activo y el texto '0' al desactivarse.
# En ambos casos invocaremos a la función mostrar_satelites con el
# texto 'sarsat.txt'.
self.sarsat = Tkinter.Checkbutton(ventana_tles, text = 'Sarsat',\
font = self.Verd11Roman, variable = self.variable_sarsat,\
onvalue = 'sarsat.txt', offvalue = 0,\
command = lambda flag = 'sarsat.txt': self.mostrar_satelites(flag))
# Fijamos la posición del objeto sarsat en el LabelFrame creado.
self.sarsat.grid(row = 1, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un nuevo objeto llamado variable_dmc del tipo StringVar.
self.variable_dmc = Tkinter.StringVar()
# Generamos un nuevo objeto llamado dmc del tipo Checkbutton.
# El texto que nos devolverá será 'dmc.txt' si se encuentra en el
# estado activo y '0' si está desactivado.
# Pasaremos la cadena de texto 'dmc.txt' a la función
# mostrar_satelites en ambos casos.
self.dmc = Tkinter.Checkbutton(ventana_tles, text = 'Dmc',\
font = self.Verd11Roman, variable = self.variable_dmc,\
onvalue = 'dmc.txt', offvalue = 0,\
command = lambda flag = 'dmc.txt': self.mostrar_satelites(flag))
# Asignamos una posición al objeto creado en su LabelFrame.
self.dmc.grid(row = 1, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un nuevo objeto del tipo StringVar llamado
```

```
# variable_tdrss.
self.variable_tdrss = Tkinter.StringVar()
# Creamos el objeto tdrss del tipo Checkbutton fijando un valor
# de respuesta para el estado activado de 'tdrss.txt' y '0' para
# el estado desactivado.
# En ambos casos pasaremos la cadena de texto 'tdrss.txt' a la
# función mostrar_satelites.
self.tdrss = Tkinter.Checkbutton(ventana_tles, text = 'Tdrss',\
font = self.Verd11Roman, variable = self.variable_tdrss,\
onvalue = 'tdrss.txt', offvalue = 0,\
command = lambda flag = 'tdrss.txt': self.mostrar_satelites(flag))
# Asignamos una posición al objeto creado dentro de su LabelFrame.
self.tdrss.grid(row = 2, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos el objeto variable_geo como un objeto StringVar.
self.variable_geo = Tkinter.StringVar()
# Generamos un objeto del tipo Checkbutton con el nombre geo.
# El texto que nos devolverá será 'geo.txt' cuando esté activo y
# '0' cuando se desactive.
# Al clickar sobre este pasaremos la cadena de texto 'geo.txt' a la
# función mostrar_satelites.
self.geo = Tkinter.Checkbutton(ventana_tles, text = 'Geo',\
font = self.Verd11Roman, variable = self.variable_geo,\
onvalue = 'geo.txt', offvalue = 0,\
command = lambda flag = 'geo.txt': self.mostrar_satelites(flag))
# Fijamos la posición del objeto geo dentro del LabelFrame
# ventana_tles.
self.geo.grid(row = 2, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un nuevo objeto del tipo StringVar con el nombre
# variable_gorizont.
self.variable_gorizont = Tkinter.StringVar()
# Creamos un objeto del tipo Checkbutton llamado ventana_tles.
# Este objeto devolverá la cadena de texto 'gorizont.txt' en el
# caso de encontrarse activo y '0' cuando esté desactivado.
# Pasará el texto 'gorizont.txt' a la función mostrar_satelites
# cuando hagamos click en él.
self.gorizont = Tkinter.Checkbutton(ventana_tles,\
text = 'Gorizont', font = self.Verd11Roman,\
variable = self.variable_gorizont, onvalue = 'gorizont.txt',\
offvalue = 0,\
command = lambda flag = 'gorizont.txt':\
self.mostrar_satelites(flag))
# Asignamos una localización al objeto gorizont dentro de su
# LabelFrame.
self.gorizont.grid(row = 2, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos el objeto variable_raduga del objeto StringVar.
self.variable_raduga = Tkinter.StringVar()
# Creamos el objeto raduga del tipo Checkbutton.
```

```
# Este objeto devolverá el texto 'raduga.txt' al activarse y '0'
# al desactivarse. En ambos casos invocará a la función
# mostrar_satelites con la cadena de caracteres 'raduga.txt'.
self.raduga = Tkinter.Checkbutton(ventana_tles, text = 'Raduga',\
font = self.Verd11Roman, variable = self.variable_raduga,\
onvalue = 'raduga.txt', offvalue = 0,\
command = lambda flag = 'raduga.txt': self.mostrar_satelites(flag))
# Fijamos la posición del objeto raduga en su localización correcta.
self.raduga.grid(row = 3, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un nuevo objeto del tipo StringVar llamado
# variable_molniya.
self.variable_molniya = Tkinter.StringVar()
# Creamos un objeto llamado molniya del tipo Checkbutton.
# Este objeto devolverá el valor 'molniya.txt' cuando esté activo
# y '0' cuando esté desactivado. En ambos casos se invocará a la
# función mostrar_satelites pasándole el texto 'molniya.txt'.
self.molniya = Tkinter.Checkbutton(ventana_tles, text = 'Molniya',\
font = self.Verd11Roman, variable = self.variable_molniya,\
onvalue = 'molniya.txt', offvalue = 0,\
command = lambda flag = 'molniya.txt': self.mostrar_satelites(flag))
# Asignamos una localización al objeto molniya dentro de su
# LabelFrame.
self.molniya.grid(row = 3, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos el objeto variable_intelsat del tipo StringVar.
self.variable_intelsat = Tkinter.StringVar()
# Creamos otro objeto del tipo Checkbutton llamado intelsat.
# Este nos devolverá la cadena de caracteres 'intelsat.txt' cuando
# se encuentre activo y '0' cuando esté desactivado.
# Al activar o desactivar el botón se invocará a la función
# mostrar_satelites con el texto 'intelsat.txt'.
self.intelsat = Tkinter.Checkbutton(ventana_tles,\
text = 'Intelsat', font = self.Verd11Roman,\
variable = self.variable_intelsat, onvalue = 'intelsat.txt',\
offvalue = 0,\
command = lambda flag = 'intelsat.txt':\
self.mostrar_satelites(flag))
# Fijamos la localización del objeto intelsat dentro de su marco.
self.intelsat.grid(row = 3, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un objeto del tipo StringVar llamado variable_iridium.
self.variable_iridium = Tkinter.StringVar()
# Generamos un objeto llamado iridium del tipo Checkbutton.
# Fijaremos un valor para el estado activado de 'iridium.txt' y
# de '0' para el estado desactivado. En ambas ocasiones llamaremos
# a la función mostrar_satelites con el texto 'iridium.txt'.
self.iridium = Tkinter.Checkbutton(ventana_tles, text = 'Iridium',\
font = self.Verd11Roman, variable = self.variable_iridium,\
onvalue = 'iridium.txt', offvalue = 0,\
```

```
command = lambda flag = 'iridium.txt': self.mostrar_satelites(flag))
# Fijaremos la posición de iridium dentro del marco ventana_tles.
self.iridium.grid(row = 4, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un objeto llamado variable_orbcomm del tipo StringVar.
self.variable_orbcomm = Tkinter.StringVar()
# Creamos el objeto orbcomm del tipo Checkbutton. Este objeto
# nos devolverá el texto 'orbcomm.txt' si se encuentra activo y el
# texto '0' si no lo está. En ambos casos, al hacer click en el
# botón, el objeto llamará a la función mostrar_satelites pasándole
# la cadena de caracteres 'orbcomm.txt'.
self.orbcomm = Tkinter.Checkbutton(ventana_tles, text = 'Orbcomm',\
font = self.Verd11Roman, variable = self.variable_orbcomm,\
onvalue = 'orbcomm.txt', offvalue = 0,\
command = lambda flag = 'orbcomm.txt': self.mostrar_satelites(flag))
# Fijamos la posición de orbcomm dentro de su marco.
self.orbcomm.grid(row = 4, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos el objeto del tipo StringVar variable_globalstar.
self.variable_globalstar = Tkinter.StringVar()
# Generamos un nuevo objeto del tipo Checkbutton llamado globalstar.
# Este nuevo objeto nos devolverá el valor 'globalstar.txt' cuando
# se encuentre activo y '0' cuando esté desactivado.
# En ambos casos el objeto llamará a la función mostrar_valores
# pasándole la cadena de texto 'globalstar.txt'.
self.globalstar = Tkinter.Checkbutton(ventana_tles,\
text = 'Globalstar', font = self.Verd11Roman,\
variable = self.variable_globalstar, onvalue = 'globalstar.txt',\
offvalue = 0,\
command = lambda flag = 'globalstar.txt':\
self.mostrar_satelites(flag))
# Asignamos una localización a globalstar dentro de su marco.
self.globalstar.grid(row = 4, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos el objeto variable_amateur del tipo StringVar.
self.variable_amateur = Tkinter.StringVar()
# Creamos el objeto amateur del tipo Checkbutton.
# Este objeto nos devolverá el valor 'amateur.txt' cuando este
# activo y '0' cuando se encuentre desactivado. Al hacer click sobre
# él se ejecutará la función mostrar_satelites con el texto
# 'amateur.txt'.
self.amateur = Tkinter.Checkbutton(ventana_tles, text = 'Amateur',\
font = self.Verd11Roman, variable = self.variable_amateur,\
onvalue = 'amateur.txt', offvalue = 0,\
command = lambda flag = 'amateur.txt':\
self.mostrar_satelites(flag))
# Fijamos la posición de amateur en el marco.
self.amateur.grid(row = 5, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)
```

```
# Generamos un nuevo objeto del tipo StringVar con el nombre
# variable_xcomm.
self.variable_xcomm = Tkinter.StringVar()
# Generamos un objeto del tipo Checkbutton llamado xcomm.
# Este Checkbutton nos devolverá el valor 'x-comm.txt' cuando esté
# activo y '0' cuando esté desactivado. Al hacer click en el objeto
# se invocará a la función mostrar_satelites con el texto
# 'x-comm.txt'.
self.xcomm = Tkinter.Checkbutton(ventana_tles, text = 'X-Comm',\
font = self.Verd11Roman, variable = self.variable_xcomm,\
onvalue = 'x-comm.txt', offvalue = 0,\
command = lambda flag = 'x-comm.txt': self.mostrar_satelites(flag))
# Asignamos una posición al objeto xcomm en su marco.
self.xcomm.grid(row = 5, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un objeto llamado variable_othercomm del tipo StringVar.
self.variable_othercomm = Tkinter.StringVar()
# Creamos un objeto llamado othercomm del tipo Checkbutton.
# Asignaremos el valor 'other-comm.txt' a othercomm cuando se
# encuentre activo y '0' cuando esté desactivado.
# En ambos casos, al hacer click sobre el objeto, se llamará a la
# función mostrar_satelites pasándole el texto 'other-comm.txt'.
self.othercomm = Tkinter.Checkbutton(ventana_tles,\
text = 'Other-Comm', font = self.Verd11Roman,\
variable = self.variable_othercomm, onvalue = 'other-comm.txt',\
offvalue = 0,\
command = lambda flag = 'other-comm.txt':\
self.mostrar_satelites(flag))
# Fijamos la posición de othercomm en el marco ventana_tles.
self.othercomm.grid(row = 5, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un nuevo objeto del tipo StringVar llamado
# variable_gpsops.
self.variable_gpsops = Tkinter.StringVar()
# Este nuevo objeto, gpsopsn nos proporcionará la cadena de texto
# 'gps-ops.txt' cuando esté activado y '0' cuando no.
# Además, al clickar sobre este se ejecutará la función
# mostrar_satelites. Para ello le daremos el texto 'gps-ops.txt'.
self.gpsops = Tkinter.Checkbutton(ventana_tles, text = 'Gps-Ops',\
font = self.Verd11Roman, variable = self.variable_gpsops,\
onvalue = 'gps-ops.txt', offvalue = 0,\
command = lambda flag = 'gps-ops.txt': self.mostrar_satelites(flag))
# Asignamos una localización a gpsops en su marco.
self.gpsops.grid(row = 6, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un objeto de tipo variable_gloops llamado variable_gloops.
self.variable_gloops = Tkinter.StringVar()
# Generamos un objeto llamado gloops del tipo Checkbutton.
# Este objeto nos devolverá la cadena de texto 'glo-ops.txt' cuando
# este activado y '0' cuando no.
```

```
# Al cambiar de estado se ejecutará la función mostrar_satelites
# con la cadena de texto 'glo-ops.txt'.
self.gloops = Tkinter.Checkbutton(ventana_tles, text = 'Glo-Ops',\
font = self.Verd11Roman, variable = self.variable_gloops,\
onvalue = 'glo-ops.txt', offvalue = 0,\
command = lambda flag = 'glo-ops.txt': self.mostrar_satelites(flag))
# Asignamos una posición a gloops dentro del marco ventana_tles.
self.gloops.grid(row = 6, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un objeto llamado variable_galileo del tipo StringVar.
self.variable_galileo = Tkinter.StringVar()
# Generamos un nuevo objeto del tipo Checkbutton con el nombre
# galileo. Este objeto nos dará la cadena de texto 'galileo.txt'
# cuando este activado y '0' cuando no.
# Al clickar sobre el mismo se ejecutará la función
# mostrar_satelites con la cadena de caracteres 'galileo.txt'.
self.galileo = Tkinter.Checkbutton(ventana_tles, text = 'Galileo',\
font = self.Verd11Roman, variable = self.variable_galileo,\
onvalue = 'galileo.txt', offvalue = 0,\
command = lambda flag = 'galileo.txt': self.mostrar_satelites(flag))
# Fijamos la posición de galileo dentro de su marco.
self.galileo.grid(row = 6, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un nuevo objeto del tipo StringVar y le damos el
# nombre variable_beidou.
self.variable_beidou = Tkinter.StringVar()
# Creamos el objeto beidou del tipo Checkbutton.
# beidou nos devolverá el texto 'beidou.txt' cuando se encuentre
# activo y '0' cuando no.
# Al clickar sobre este objeto invocaremos a la función
# mostrar_satelites con la cadena de caracteres beidou.txt'.
self.beidou = Tkinter.Checkbutton(ventana_tles, text = 'Beidou',\
font = self.Verd11Roman, variable = self.variable_beidou,\
onvalue = 'beidou.txt', offvalue = 0,\
command = lambda flag = 'beidou.txt': self.mostrar_satelites(flag))
# Fijamos la posición de beidou.
self.beidou.grid(row = 7, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un nuevo objeto llamado variable_sbas del tipo
# StringVar.
self.variable_sbas = Tkinter.StringVar()
# Creamos un objeto del tipo Checkbutton llamado sbas.
# Este objeto, sbas, nos devolverá la cadena de texto 'sbas.txt'
# cuando esté activo y '0' cuando esté desactivado.
# Además, al cambiar entre uno y otro estado, se ejecutará la
# función mostrar_satelites. Para ello le pasaremos la cadena
# de caracteres 'sbas.txt'.
self.sbas = Tkinter.Checkbutton(ventana_tles, text = 'Sbas',\
font = self.Verd11Roman, variable = self.variable_sbas,\
onvalue = 'sbas.txt', offvalue = 0,\
```

```
command = lambda flag = 'sbas.txt': self.mostrar_satelites(flag))
# Asignamos una posición al objeto sbas dentro del marco
# ventana_tles.
self.sbas.grid(row = 7, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un objeto del tipo StringVar llamado variable_nnss.
self.variable_nnss = Tkinter.StringVar()
# Generamos un nuevo objeto del tipo Checkbutton con el nombre nnss.
# Este objeto nos devolverá la cadena de texto 'nnss.txt' cuando
# se encuentra activo y '0' cuando no.
# Al clickar sobre el objeto se ejecutará la función
# mostrar_satelites pasándole el texto 'nnss.txt'.
self.nnss = Tkinter.Checkbutton(ventana_tles, text = 'Nnss',\
font = self.Verd11Roman, variable = self.variable_nnss,\
onvalue = 'nnss.txt', offvalue = 0,\
command = lambda flag = 'nnss.txt': self.mostrar_satelites(flag))
# Fijamos la posición del objeto nnss en su marco.
self.nnss.grid(row = 7, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos el objeto variable_musson del tipo StringVar.
self.variable_musson = Tkinter.StringVar()
# Creamos un nuevo objeto del tipo Checkbutton llamado musson.
# Este objeto nos devolverá el texto 'musson.txt' cuando se
# encuentre activo y '0' cuando no sea así.
# En ambos casos, al cambiar de estado, se ejecutará la función
# mostrar_satelites pasándole el texto 'musson.txt'.
self.musson = Tkinter.Checkbutton(ventana_tles, text = 'Musson',\
font = self.Verd11Roman, variable = self.variable_musson,\
onvalue = 'musson.txt', offvalue = 0,\
command = lambda flag = 'musson.txt': self.mostrar_satelites(flag))
# Asignamos una posición al objeto musson dentro del
# marco ventana_tles.
self.musson.grid(row = 8, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un objeto StringVar llamado variable_science.
self.variable_science = Tkinter.StringVar()
# Generamos un nuevo objeto del tipo Checkbutton dentro del marco
# ventana_tles.
# Este objeto nos proporcionará el texto 'science.txt' cuando este
# activado y '0' cuando no sea así. Además, al hacer click sobre él,
# se invocará la función mostrar_tles con el texto 'science.txt'.
self.science = Tkinter.Checkbutton(ventana_tles, text = 'Science',\
font = self.Verd11Roman, variable = self.variable_science,\
onvalue = 'science.txt', offvalue = 0,\
command = lambda flag = 'science.txt':\
self.mostrar_satelites(flag))
# Fijamos la posición de science dentro de su marco.
self.science.grid(row = 8, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)
```

```
# Generamos el objeto variable_geodetic del tipo StringVar.
self.variable_geodetic = Tkinter.StringVar()
# Creamos el objeto geodetic del tipo Checkbutton.
# Este objeto nos devolverá la cadena de texto 'geodetic.txt'
# cuando este activo y '0' cuando no. Al cambiar de estado este
# objeto invocará a la función mostrar_valores pasándole la cadena
# de texto 'geodetic.txt'.
self.geodetic = Tkinter.Checkbutton(ventana_tles,\
text = 'Geodetic', font = self.Verd11Roman,\
variable = self.variable_geodetic, onvalue = 'geodetic.txt',\
offvalue = 0,\
command = lambda flag = 'geodetic.txt':\
self.mostrar_satelites(flag))
# Asignamos la posición de geodetic en el marco ventana_tles.
self.geodetic.grid(row = 8, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos el objeto de tipo StringVar llamado variable_engineering.
self.variable_engineering = Tkinter.StringVar()
# Generamos un nuevo objeto llamado engineering de tipo Checkbutton.
# Este objeto nos devolverá el valor 'engineering.txt' cuando se
# encuentre activo y '0' cuando no.
# Al clickar sobre este ejecutaremos la función mostrar_satelites
# pasándole la cadena de texto 'engineering.txt'.
self.engineering = Tkinter.Checkbutton(ventana_tles,\
text = 'Engineering', font = self.Verd11Roman,\
variable = self.variable_engineering, onvalue = 'engineering.txt',\
offvalue = 0,\
command = lambda flag = 'engineering.txt':\
self.mostrar_satelites(flag))
# Asignamos la posición de engineering en su marco.
self.engineering.grid(row = 9, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Crearemos un objeto del tipo StringVar llamado variable_education.
self.variable_education = Tkinter.StringVar()
# Generamos un objeto llamado education del tipo Checkbutton.
# Este objeto nos dará la cadena de texto 'education.txt' cuando
# se encuentra activo y '0' cuando no.
# Al cambiar de estado llamaremos a la función mostrar_satelites
# pasándole el texto 'education.txt'.
self.education = Tkinter.Checkbutton(ventana_tles,\
text = 'Education', font = self.Verd11Roman,\
variable = self.variable_education, onvalue = 'education.txt',\
offvalue = 0,\
command = lambda flag = 'education.txt':\
self.mostrar_satelites(flag))
# Fijamos la posición de education en el marco ventana_tles.
self.education.grid(row = 9, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos el objeto variable_military del tipo StringVar.
self.variable_military = Tkinter.StringVar()
```

```
# A continuación generamos un nuevo objeto del tipo Checkbutton
# llamado military. Este objeto nos devolverá la cadena de texto
# 'military.txt' cuando se encuentra activo y '0' cuando no lo está.
# Al clickar sobre el botón llamamos a la rutina mostrar_satelites
# con el texto 'military.txt'.
self.military = Tkinter.Checkbutton(ventana_tles,\
text = 'Military', font = self.Verd11Roman,\
variable = self.variable_military, onvalue = 'military.txt',\
offvalue = 0,\
command = lambda flag = 'military.txt':\
self.mostrar_satelites(flag))
# Fijamos la posición de military dentro del marco ventana_tles.
self.military.grid(row = 9, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un objeto llamado variable_radar del tipo StringVar.
self.variable_radar = Tkinter.StringVar()
# Generamos el objeto radar del tipo Checkbutton.
# Este objeto nos devolverá la cadena de texto 'radar.txt' cuando
# este activo y '0' cuando no lo esté.
# Al cambiar de estado se ejecutará la función mostrar_satelites
# con el texto 'radar.txt'.
self.radar = Tkinter.Checkbutton(ventana_tles, text = 'Radar',\
font = self.Verd11Roman, variable = self.variable_radar,\
onvalue = 'radar.txt', offvalue = 0,\
command = lambda flag = 'radar.txt': self.mostrar_satelites(flag))
# Fijamos la posición del objeto radar dentro de su marco.
self.radar.grid(row = 10, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un nuevo objeto del tipo StringVar llamado
# variable_cubesat.
self.variable_cubesat = Tkinter.StringVar()
# Generamos un objeto llamado cubesat del tipo Checkbutton.
# El valor de respuesta del objeto será 'cubesat.txt' cuando esté
# activado y '0' cuando esté desactivado.
# Invocaremos a la función mostrar_satelites con el texto
# 'cubesat.txt' cuando cambie de estado.
self.cubesat = Tkinter.Checkbutton(ventana_tles, text = 'Cubesat',\
font = self.Verd11Roman, variable = self.variable_cubesat,\
onvalue = 'cubesat.txt', offvalue = 0,\
command = lambda flag = 'cubesat.txt': self.mostrar_satelites(flag))
# Asignamos una localización al objeto dentro de su marco.
self.cubesat.grid(row = 10, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un nuevo objeto del tipo StringVar llamado
# variable_other.
self.variable_other = Tkinter.StringVar()
# Creamos un objeto llamado other del tipo Checkbutton.
# Este objeto nos devolverá el valor 'other.txt' cuando esté
# activo y '0' cuando esté desactivado. En ambos casos, al hacer
# click sobre el mismo, se ejecutará la función mostrar_satelites
```

```
        # con el texto 'other.txt'.
        self.other = Tkinter.Checkbutton(ventana_tles, text = 'Other',\
        font = self.Verd11Roman, variable = self.variable_other,\
        onvalue = 'other.txt', offvalue = 0,\
        command = lambda flag = 'other.txt': self.mostrar_satelites(flag))
        # Fijamos la posición de other dentro del marco ventana_tles.
        self.other.grid(row = 10, column = 2, rowspan = 1,\
        columnspan = 1, sticky = Tkinter.W)

    elif sys.platform.startswith('linux'):
        # A continuación crearemos cada Checkbutton. El proceso de creación
        # de cada una de ellos será muy similar. Primero generamos un
        # nuevo objeto del tipo StringVar. Ya que la cadena de texto que
        # devolverán los Checkbutton será distinta dependiendo de si están
        # seleccionados o no necesitaremos un objeto de texto que cambie
        # de valor dinamicamente.
        # Hecho esto generamos el objeto de tipo Checkbutton asignando los
        # valores de activado y desactivado correspondientes y colocando
        # el objeto en su lugar.
        # También vincularemos nuestro objeto a la función mostrar_satelites
        # lo cual nos permitirá mostrar los satélites disponibles en cada
        # familia en una lista del tipo ScrolledList.

        # Creación del objeto de tipo StringVar variable_weather.
        self.variable_weather = Tkinter.StringVar()
        # Creación del objeto de tipo Checkbutton weather.
        # Definimos como valor para cuando este activado 'weather.txt' y '0'
        # para cuando no esté activado.
        # Al pulsar el objeto pasaremos la cadena de caracteres
        # 'weather.txt' a la función mostrar_satelites.
        self.weather = Tkinter.Checkbutton(ventana_tles, text = 'Weather',\
        font = self.Verd11Roman, variable = self.variable_weather,\
        onvalue = 'weather.txt', offvalue = 0,\
        command = lambda flag = 'weather.txt': self.mostrar_satelites(flag))
        # Asignamos una posición al objeto creado.
        self.weather.grid(row = 0, column = 0, rowspan = 1,\
        columnspan = 1, sticky = Tkinter.W)

        # Generamos el objeto variable_noaa del tipo StringVar.
        self.variable_noaa = Tkinter.StringVar()
        # Creamos un Checkbutton llamado noaa. El valor que nos devolverá
        # al activarse será 'noaa.txt' devolviéndonos '0' al desactivarse.
        # Invocaremos a la función mostrar_satelites con el texto
        # 'noaa.txt' al pulsar sobre el objeto.
        self.noaa = Tkinter.Checkbutton(ventana_tles, text = 'Noaa',\
        font = self.Verd11Roman, variable = self.variable_noaa,\
        onvalue = 'noaa.txt', offvalue = 0,\
        command = lambda flag = 'noaa.txt': self.mostrar_satelites(flag))
        # Asignamos una posición al objeto en la ventana.
        self.noaa.grid(row = 0, column = 1, rowspan = 1,\
        columnspan = 1, sticky = Tkinter.W)

        # Generamos un objeto variable_goes del tipo StringVar.
```

```
self.variable_goes = Tkinter.StringVar()
# Crearemos un objeto llamado goes del tipo Checkbutton.
# Este objeto nos devolverá la cadena de caracteres 'goes.txt'
# al activar el objeto y '0' al desactivarlo.
# Al pulsarlo, en cualquiera de las ocasiones, llamaremos a la
# función mostrar_satelites con el texto 'goes.txt'.
self.goes = Tkinter.Checkbutton(ventana_tles, text = 'Goes',\
font = self.Verd11Roman, variable = self.variable_goes,\
onvalue = 'goes.txt', offvalue = 0,\
command = lambda flag = 'goes.txt': self.mostrar_satelites(flag))
# Fijamos una posición para el objeto en el LabelFrame ventana_tles.
self.goes.grid(row = 0, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un objeto del tipo StringVar llamado variable_resource.
self.variable_resource = Tkinter.StringVar()
# Generamos un nuevo objeto del tipo Checkbutton. El texto que nos
# devolverá al activarlo será 'resource.txt' siendo '0' al
# desactivarlo.
# Al hacer click en el objeto pasaremos el texto 'resource.txt' a
# la función mostrar_satelites.
self.resource = Tkinter.Checkbutton(ventana_tles,\
text = 'Resource', font = self.Verd11Roman,\
variable = self.variable_resource, onvalue = 'resource.txt',\
offvalue = 0,\
command = lambda flag = 'resource.txt':\
self.mostrar_satelites(flag))
# Asignamos una posición al objeto creado en el objeto LabelFrame.
self.resource.grid(row = 1, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un nuevo objeto del tipo StringVar llamado
# variable_sarsat.
self.variable_sarsat = Tkinter.StringVar()
# Creamos un objeto llamado sarsat del tipo Checkbutton.
# Este objeto nos devolverá el texto 'sarsat.txt' cuando se
# encuentre activo y el texto '0' al desactivarse.
# En ambos casos invocaremos a la función mostrar_satelites con el
# texto 'sarsat.txt'.
self.sarsat = Tkinter.Checkbutton(ventana_tles, text = 'Sarsat',\
font = self.Verd11Roman, variable = self.variable_sarsat,\
onvalue = 'sarsat.txt', offvalue = 0,\
command = lambda flag = 'sarsat.txt': self.mostrar_satelites(flag))
# Fijamos la posición del objeto sarsat en el LabelFrame creado.
self.sarsat.grid(row = 1, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un nuevo objeto llamado variable_dmc del tipo StringVar.
self.variable_dmc = Tkinter.StringVar()
# Generamos un nuevo objeto llamado dmc del tipo Checkbutton.
# El texto que nos devolverá será 'dmc.txt' si se encuentra en el
# estado activo y '0' si está desactivado.
# Pasaremos la cadena de texto 'dmc.txt' a la función
```

```
# mostrar_satelites en ambos casos.
self.dmc = Tkinter.Checkbutton(ventana_tles, text = 'Dmc',\
font = self.Verd11Roman, variable = self.variable_dmc,\
onvalue = 'dmc.txt', offvalue = 0,\
command = lambda flag = 'dmc.txt': self.mostrar_satelites(flag))
# Asignamos una posición al objeto creado en su LabelFrame.
self.dmc.grid(row = 1, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un nuevo objeto del tipo StringVar llamado
# variable_tdrss.
self.variable_tdrss = Tkinter.StringVar()
# Creamos el objeto tdrss del tipo Checkbutton fijando un valor
# de respuesta para el estado activado de 'tdrss.txt' y '0' para
# el estado desactivado.
# En ambos casos pasaremos la cadena de texto 'tdrss.txt' a la
# función mostrar_satelites.
self.tdrss = Tkinter.Checkbutton(ventana_tles, text = 'Tdrss',\
font = self.Verd11Roman, variable = self.variable_tdrss,\
onvalue = 'tdrss.txt', offvalue = 0,\
command = lambda flag = 'tdrss.txt': self.mostrar_satelites(flag))
# Asignamos una posición al objeto creado dentro de su LabelFrame.
self.tdrss.grid(row = 2, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos el objeto variable_geo como un objeto StringVar.
self.variable_geo = Tkinter.StringVar()
# Generamos un objeto del tipo Checkbutton con el nombre geo.
# El texto que nos devolverá será 'geo.txt' cuando esté activo y
# '0' cuando se desactive.
# Al clickar sobre este pasaremos la cadena de texto 'geo.txt' a la
# función mostrar_satelites.
self.geo = Tkinter.Checkbutton(ventana_tles, text = 'Geo',\
font = self.Verd11Roman, variable = self.variable_geo,\
onvalue = 'geo.txt', offvalue = 0,\
command = lambda flag = 'geo.txt': self.mostrar_satelites(flag))
# Fijamos la posición del objeto geo dentro del LabelFrame
# ventana_tles.
self.geo.grid(row = 2, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un nuevo objeto del tipo StringVar con el nombre
# variable_gorizont.
self.variable_gorizont = Tkinter.StringVar()
# Creamos un objeto del tipo Checkbutton llamado ventana_tles.
# Este objeto devolverá la cadena de texto 'gorizont.txt' en el
# caso de encontrarse activo y '0' cuando esté desactivado.
# Pasará el texto 'gorizont.txt' a la función mostrar_satelites
# cuando hagamos click en él.
self.gorizont = Tkinter.Checkbutton(ventana_tles,\
text = 'Gorizont', font = self.Verd11Roman,\
variable = self.variable_gorizont, onvalue = 'gorizont.txt',\
offvalue = 0,\
```

```
command = lambda flag = 'gorizont.txt':\
self.mostrar_satelites(flag))
# Asignamos una localización al objeto gorizont dentro de su
# LabelFrame.
self.gorizont.grid(row = 2, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos el objeto variable_raduga del objeto StringVar.
self.variable_raduga = Tkinter.StringVar()
# Creamos el objeto raduga del tipo Checkbutton.
# Este objeto devolverá el texto 'raduga.txt' al activarse y '0'
# al desactivarse. En ambos casos invocará a la función
# mostrar_satelites con la cadena de caracteres 'raduga.txt'.
self.raduga = Tkinter.Checkbutton(ventana_tles, text = 'Raduga',\
font = self.Verd11Roman, variable = self.variable_raduga,\
onvalue = 'raduga.txt', offvalue = 0,\
command = lambda flag = 'raduga.txt': self.mostrar_satelites(flag))
# Fijamos la posición del objeto raduga en su localización correcta.
self.raduga.grid(row = 3, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un nuevo objeto del tipo StringVar llamado
# variable_molniya.
self.variable_molniya = Tkinter.StringVar()
# Creamos un objeto llamado molniya del tipo Checkbutton.
# Este objeto devolverá el valor 'molniya.txt' cuando esté activo
# y '0' cuando esté desactivado. En ambos casos se invocará a la
# función mostrar_satelites pasándole el texto 'molniya.txt'.
self.molniya = Tkinter.Checkbutton(ventana_tles, text = 'Molniya',\
font = self.Verd11Roman, variable = self.variable_molniya,\
onvalue = 'molniya.txt', offvalue = 0,\
command = lambda flag = 'molniya.txt': self.mostrar_satelites(flag))
# Asignamos una localización al objeto molniya dentro de su
# LabelFrame.
self.molniya.grid(row = 3, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos el objeto variable_intelsat del tipo StringVar.
self.variable_intelsat = Tkinter.StringVar()
# Creamos otro objeto del tipo Checkbutton llamado intelsat.
# Este nos devolverá la cadena de caracteres 'intelsat.txt' cuando
# se encuentre activo y '0' cuando esté desactivado.
# Al activar o desactivar el botón se invocará a la función
# mostrar_satelites con el texto 'intelsat.txt'.
self.intelsat = Tkinter.Checkbutton(ventana_tles,\
text = 'Intelsat', font = self.Verd11Roman,\
variable = self.variable_intelsat, onvalue = 'intelsat.txt',\
offvalue = 0,\
command = lambda flag = 'intelsat.txt':\
self.mostrar_satelites(flag))
# Fijamos la localización del objeto intelsat dentro de su marco.
self.intelsat.grid(row = 3, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)
```

```
# Generamos un objeto del tipo StringVar llamado variable_iridium.
self.variable_iridium = Tkinter.StringVar()
# Generamos un objeto llamado iridium del tipo Checkbutton.
# Fijaremos un valor para el estado activado de 'iridium.txt' y
# de '0' para el estado desactivado. En ambas ocasiones llamaremos
# a la función mostrar_satelites con el texto 'iridium.txt'.
self.iridium = Tkinter.Checkbutton(ventana_tles, text = 'Iridium',\
font = self.Verd11Roman, variable = self.variable_iridium,\
onvalue = 'iridium.txt', offvalue = 0,\
command = lambda flag = 'iridium.txt': self.mostrar_satelites(flag))
# Fijaremos la posición de iridium dentro del marco ventana_tles.
self.iridium.grid(row = 4, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un objeto llamado variable_orbcomm del tipo StringVar.
self.variable_orbcomm = Tkinter.StringVar()
# Creamos el objeto orbcomm del tipo Checkbutton. Este objeto
# nos devolverá el texto 'orbcomm.txt' si se encuentra activo y el
# texto '0' si no lo está. En ambos casos, al hacer click en el
# botón, el objeto llamará a la función mostrar_satelites pasándole
# la cadena de caracteres 'orbcomm.txt'.
self.orbcomm = Tkinter.Checkbutton(ventana_tles, text = 'Orbcomm',\
font = self.Verd11Roman, variable = self.variable_orbcomm,\
onvalue = 'orbcomm.txt', offvalue = 0,\
command = lambda flag = 'orbcomm.txt': self.mostrar_satelites(flag))
# Fijamos la posición de orbcomm dentro de su marco.
self.orbcomm.grid(row = 4, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos el objeto del tipo StringVar variable_globalstar.
self.variable_globalstar = Tkinter.StringVar()
# Generamos un nuevo objeto del tipo Checkbutton llamado globalstar.
# Este nuevo objeto nos devolverá el valor 'globalstar.txt' cuando
# se encuentre activo y '0' cuando esté desactivado.
# En ambos casos el objeto llamará a la función mostrar_valores
# pasándole la cadena de texto 'globalstar.txt'.
self.globalstar = Tkinter.Checkbutton(ventana_tles,\
text = 'Globalstar', font = self.Verd11Roman,\
variable = self.variable_globalstar, onvalue = 'globalstar.txt',\
offvalue = 0,\
command = lambda flag = 'globalstar.txt':\
self.mostrar_satelites(flag))
# Asignamos una localización a globalstar dentro de su marco.
self.globalstar.grid(row = 4, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos el objeto variable_amateur del tipo StringVar.
self.variable_amateur = Tkinter.StringVar()
# Creamos el objeto amateur del tipo Checkbutton.
# Este objeto nos devolverá el valor 'amateur.txt' cuando este
# activo y '0' cuando se encuentre desactivado. Al hacer click sobre
# él se ejecutará la función mostrar_satelites con el texto
```

```
# 'amateur.txt'.
self.amateur = Tkinter.Checkbutton(ventana_tles, text = 'Amateur',\
font = self.Verd11Roman, variable = self.variable_amateur,\
onvalue = 'amateur.txt', offvalue = 0,\
command = lambda flag = 'amateur.txt': self.mostrar_satelites(flag))
# Fijamos la posición de amateur en el marco.
self.amateur.grid(row = 5, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un nuevo objeto del tipo StringVar con el nombre
# variable_xcomm.
self.variable_xcomm = Tkinter.StringVar()
# Generamos un objeto del tipo Checkbutton llamado xcomm.
# Este Checkbutton nos devolverá el valor 'x-comm.txt' cuando esté
# activo y '0' cuando esté desactivado. Al hacer click en el objeto
# se invocará a la función mostrar_satelites con el texto
# 'x-comm.txt'.
self.xcomm = Tkinter.Checkbutton(ventana_tles, text = 'X-Comm',\
font = self.Verd11Roman, variable = self.variable_xcomm,\
onvalue = 'x-comm.txt', offvalue = 0,\
command = lambda flag = 'x-comm.txt': self.mostrar_satelites(flag))
# Asignamos una posición al objeto xcomm en su marco.
self.xcomm.grid(row = 5, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un objeto llamado variable_othercomm del tipo StringVar.
self.variable_othercomm = Tkinter.StringVar()
# Creamos un objeto llamado othercomm del tipo Checkbutton.
# Asignaremos el valor 'other-comm.txt' a othercomm cuando se
# encuentre activo y '0' cuando esté desactivado.
# En ambos casos, al hacer click sobre el objeto, se llamará a la
# función mostrar_satelites pasándole el texto 'other-comm.txt'.
self.othercomm = Tkinter.Checkbutton(ventana_tles,\
text = 'Other-Comm', font = self.Verd11Roman,\
variable = self.variable_othercomm, onvalue = 'other-comm.txt',\
offvalue = 0,\
command = lambda flag = 'other-comm.txt':\
self.mostrar_satelites(flag))
# Fijamos la posición de othercomm en el marco ventana_tles.
self.othercomm.grid(row = 5, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un nuevo objeto del tipo StringVar llamado
# variable_gpsops.
self.variable_gpsops = Tkinter.StringVar()
# Este nuevo objeto, gpsopsn nos proporcionará la cadena de texto
# 'gps-ops.txt' cuando esté activado y '0' cuando no.
# Además, al clickar sobre este se ejecutará la función
# mostrar_satelites. Para ello le daremos el texto 'gps-ops.txt'.
self.gpsops = Tkinter.Checkbutton(ventana_tles, text = 'Gps-Ops',\
font = self.Verd11Roman, variable = self.variable_gpsops,\
onvalue = 'gps-ops.txt', offvalue = 0,\
command = lambda flag = 'gps-ops.txt': self.mostrar_satelites(flag))
```

```
# Asignamos una localización a gpsops en su marco.
self.gpsops.grid(row = 6, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un objeto de tipo variable_gloops llamado variable_gloops.
self.variable_gloops = Tkinter.StringVar()
# Generamos un objeto llamado gloops del tipo Checkbutton.
# Este objeto nos devolverá la cadena de texto 'glo-ops.txt' cuando
# este activado y '0' cuando no.
# Al cambiar de estado se ejecutará la función mostrar_satelites
# con la cadena de texto 'glo-ops.txt'.
self.gloops = Tkinter.Checkbutton(ventana_tles, text = 'Glo-Ops',\
font = self.Verd11Roman, variable = self.variable_gloops,\
onvalue = 'glo-ops.txt', offvalue = 0,\
command = lambda flag = 'glo-ops.txt': self.mostrar_satelites(flag))
# Asignamos una posición a gloops dentro del marco ventana_tles.
self.gloops.grid(row = 6, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un objeto llamado variable_galileo del tipo StringVar.
self.variable_galileo = Tkinter.StringVar()
# Generamos un nuevo objeto del tipo Checkbutton con el nombre
# galileo. Este objeto nos dará la cadena de texto 'galileo.txt'
# cuando este activado y '0' cuando no.
# Al clickar sobre el mismo se ejecutará la función
# mostrar_satelites con la cadena de caracteres 'galileo.txt'.
self.galileo = Tkinter.Checkbutton(ventana_tles, text = 'Galileo',\
font = self.Verd11Roman, variable = self.variable_galileo,\
onvalue = 'galileo.txt', offvalue = 0,\
command = lambda flag = 'galileo.txt': self.mostrar_satelites(flag))
# Fijamos la posición de galileo dentro de su marco.
self.galileo.grid(row = 6, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un nuevo objeto del tipo StringVar y le damos el
# nombre variable_beidou.
self.variable_beidou = Tkinter.StringVar()
# Creamos el objeto beidou del tipo Checkbutton.
# beidou nos devolverá el texto 'beidou.txt' cuando se encuentre
# activo y '0' cuando no.
# Al clickar sobre este objeto invocaremos a la función
# mostrar_satelites con la cadena de caracteres beidou.txt'.
self.beidou = Tkinter.Checkbutton(ventana_tles, text = 'Beidou',\
font = self.Verd11Roman, variable = self.variable_beidou,\
onvalue = 'beidou.txt', offvalue = 0,\
command = lambda flag = 'beidou.txt': self.mostrar_satelites(flag))
# Fijamos la posición de beidou.
self.beidou.grid(row = 7, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un nuevo objeto llamado variable_sbas del tipo
# StringVar.
self.variable_sbas = Tkinter.StringVar()
```

```
# Creamos un objeto del tipo Checkbutton llamado sbas.
# Este objeto, sbas, nos devolverá la cadena de texto 'sbas.txt'
# cuando esté activo y '0' cuando esté desactivado.
# Además, al cambiar entre uno y otro estado, se ejecutará la
# función mostrar_satelites. Para ello le pasaremos la cadena
# de caracteres 'sbas.txt'.
self.sbas = Tkinter.Checkbutton(ventana_tles, text = 'Sbas',\
font = self.Verd11Roman, variable = self.variable_sbas,\
onvalue = 'sbas.txt', offvalue = 0,\
command = lambda flag = 'sbas.txt': self.mostrar_satelites(flag))
# Asignamos una posición al objeto sbas dentro del marco
# ventana_tles.
self.sbas.grid(row = 7, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un objeto del tipo StringVar llamado variable_nnss.
self.variable_nnss = Tkinter.StringVar()
# Generamos un nuevo objeto del tipo Checkbutton con el nombre nnss.
# Este objeto nos devolverá la cadena de texto 'nnss.txt' cuando
# se encuentra activo y '0' cuando no.
# Al clickar sobre el objeto se ejecutará la función
# mostrar_satelites pasándole el texto 'nnss.txt'.
self.nnss = Tkinter.Checkbutton(ventana_tles, text = 'Nnss',\
font = self.Verd11Roman, variable = self.variable_nnss,\
onvalue = 'nnss.txt', offvalue = 0,\
command = lambda flag = 'nnss.txt': self.mostrar_satelites(flag))
# Fijamos la posición del objeto nnss en su marco.
self.nnss.grid(row = 7, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos el objeto variable_musson del tipo StringVar.
self.variable_musson = Tkinter.StringVar()
# Creamos un nuevo objeto del tipo Checkbutton llamado musson.
# Este objeto nos devolverá el texto 'musson.txt' cuando se
# encuentre activo y '0' cuando no sea así.
# En ambos casos, al cambiar de estado, se ejecutará la función
# mostrar_satelites pasándole el texto 'musson.txt'.
self.musson = Tkinter.Checkbutton(ventana_tles, text = 'Musson',\
font = self.Verd11Roman, variable = self.variable_musson,\
onvalue = 'musson.txt', offvalue = 0,\
command = lambda flag = 'musson.txt': self.mostrar_satelites(flag))
# Asignamos una posición al objeto musson dentro del
# marco ventana_tles.
self.musson.grid(row = 8, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un objeto StringVar llamado variable_science.
self.variable_science = Tkinter.StringVar()
# Generamos un nuevo objeto del tipo Checkbutton dentro del marco
# ventana_tles.
# Este objeto nos proporcionará el texto 'science.txt' cuando este
# activado y '0' cuando no sea así. Además, al hacer click sobre él,
# se invocará la función mostrar_tles con el texto 'science.txt'.
```

```
self.science = Tkinter.Checkbutton(ventana_tles, text = 'Science',\
font = self.Verd11Roman, variable = self.variable_science,\
onvalue = 'science.txt', offvalue = 0,\
command = lambda flag = 'science.txt': self.mostrar_satelites(flag))
# Fijamos la posición de science dentro de su marco.
self.science.grid(row = 8, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos el objeto variable_geodetic del tipo StringVar.
self.variable_geodetic = Tkinter.StringVar()
# Creamos el objeto geodetic del tipo Checkbutton.
# Este objeto nos devolverá la cadena de texto 'geodetic.txt'
# cuando este activo y '0' cuando no. Al cambiar de estado este
# objeto invocará a la función mostrar_valores pasándole la cadena
# de texto 'geodetic.txt'.
self.geodetic = Tkinter.Checkbutton(ventana_tles,\
text = 'Geodetic', font = self.Verd11Roman,\
variable = self.variable_geodetic, onvalue = 'geodetic.txt',\
offvalue = 0,\
command = lambda flag = 'geodetic.txt':\
self.mostrar_satelites(flag))
# Asignamos la posición de geodetic en el marco ventana_tles.
self.geodetic.grid(row = 8, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos el objeto de tipo StringVar llamado variable_engineering.
self.variable_engineering = Tkinter.StringVar()
# Generamos un nuevo objeto llamado engineering de tipo Checkbutton.
# Este objeto nos devolverá el valor 'engineering.txt' cuando se
# encuentre activo y '0' cuando no.
# Al clickar sobre este ejecutaremos la función mostrar_satelites
# pasándole la cadena de texto 'engineering.txt'.
self.engineering = Tkinter.Checkbutton(ventana_tles,\
text = 'Engineering', font = self.Verd11Roman,\
variable = self.variable_engineering, onvalue = 'engineering.txt',\
offvalue = 0,\
command = lambda flag = 'engineering.txt':\
self.mostrar_satelites(flag))
# Asignamos la posición de engineering en su marco.
self.engineering.grid(row = 9, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Crearemos un objeto del tipo StringVar llamado variable_education.
self.variable_education = Tkinter.StringVar()
# Generamos un objeto llamado education del tipo Checkbutton.
# Este objeto nos dará la cadena de texto 'education.txt' cuando
# se encuentra activo y '0' cuando no.
# Al cambiar de estado llamaremos a la función mostrar_satelites
# pasándole el texto 'education.txt'.
self.education = Tkinter.Checkbutton(ventana_tles,\
text = 'Education', font = self.Verd11Roman,\
variable = self.variable_education, onvalue = 'education.txt',\
offvalue = 0,\
```

```
command = lambda flag = 'education.txt':\
self.mostrar_satelites(flag))
# Fijamos la posición de education en el marco ventana_tles.
self.education.grid(row = 9, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos el objeto variable_military del tipo StringVar.
self.variable_military = Tkinter.StringVar()
# A continuación generamos un nuevo objeto del tipo Checkbutton
# llamado military. Este objeto nos devolverá la cadena de texto
# 'military.txt' cuando se encuentra activo y '0' cuando no lo está.
# Al clickar sobre el botón llamamos a la rutina mostrar_satelites
# con el texto 'military.txt'.
self.military = Tkinter.Checkbutton(ventana_tles,\
text = 'Military', font = self.Verd11Roman,\
variable = self.variable_military, onvalue = 'military.txt',\
offvalue = 0,\
command = lambda flag = 'military.txt':\
self.mostrar_satelites(flag))
# Fijamos la posición de military dentro del marco ventana_tles.
self.military.grid(row = 9, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un objeto llamado variable_radar del tipo StringVar.
self.variable_radar = Tkinter.StringVar()
# Generamos el objeto radar del tipo Checkbutton.
# Este objeto nos devolverá la cadena de texto 'radar.txt' cuando
# este activo y '0' cuando no lo esté.
# Al cambiar de estado se ejecutará la función mostrar_satelites
# con el texto 'radar.txt'.
self.radar = Tkinter.Checkbutton(ventana_tles, text = 'Radar',\
font = self.Verd11Roman, variable = self.variable_radar,\
onvalue = 'radar.txt', offvalue = 0,\
command = lambda flag = 'radar.txt': self.mostrar_satelites(flag))
# Fijamos la posición del objeto radar dentro de su marco.
self.radar.grid(row = 10, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un nuevo objeto del tipo StringVar llamado
# variable_cubesat.
self.variable_cubesat = Tkinter.StringVar()
# Generamos un objeto llamado cubesat del tipo Checkbutton.
# El valor de respuesta del objeto será 'cubesat.txt' cuando esté
# activado y '0' cuando esté desactivado.
# Invocaremos a la función mostrar_satelites con el texto
# 'cubesat.txt' cuando cambie de estado.
self.cubesat = Tkinter.Checkbutton(ventana_tles, text = 'Cubesat',\
font = self.Verd11Roman, variable = self.variable_cubesat,\
onvalue = 'cubesat.txt', offvalue = 0,\
command = lambda flag = 'cubesat.txt':\
self.mostrar_satelites(flag))
# Asignamos una localización al objeto dentro de su marco.
self.cubesat.grid(row = 10, column = 1, rowspan = 1,\
```

```
            columnspan = 1, sticky = Tkinter.W)

            # Creamos un nuevo objeto del tipo StringVar llamado
            # variable_other.
            self.variable_other = Tkinter.StringVar()
            # Creamos un objeto llamado other del tipo Checkbutton.
            # Este objeto nos devolverá el valor 'other.txt' cuando esté
            # activo y '0' cuando esté desactivado. En ambos casos, al hacer
            # click sobre el mismo, se ejecutará la función mostrar_satelites
            # con el texto 'other.txt'.
            self.other = Tkinter.Checkbutton(ventana_tles, text = 'Other',\
            font = self.Verd11Roman, variable = self.variable_other,\
            onvalue = 'other.txt', offvalue = 0,\
            command = lambda flag = 'other.txt': self.mostrar_satelites(flag))
            # Fijamos la posición de other dentro del marco ventana_tles.
            self.other.grid(row = 10, column = 2, rowspan = 1,\
            columnspan = 1, sticky = Tkinter.W)

        elif sys.platform == "win32":
            # A continuación crearemos cada Checkbutton. El proceso de creación
            # de cada una de ellos será muy similar. Primero generamos un
            # nuevo objeto del tipo StringVar. Ya que la cadena de texto que
            # devolverán los Checkbutton será distinta dependiendo de si están
            # seleccionados o no necesitaremos un objeto de texto que cambie
            # de valor dinamicamente.
            # Hecho esto generamos el objeto de tipo Checkbutton asignando los
            # valores de activado y desactivado correspondientes y colocando
            # el objeto en su lugar.
            # También vincularemos nuestro objeto a la función mostrar_satelites
            # lo cual nos permitirá mostrar los satélites disponibles en cada
            # familia en una lista del tipo ScrolledList.

            # Creación del objeto de tipo StringVar variable_weather.
            self.variable_weather = Tkinter.StringVar()
            # Creación del objeto de tipo Checkbutton weather.
            # Definimos como valor para cuando este activado 'weather.txt' y '0'
            # para cuando no esté activado.
            # Al pulsar el objeto pasaremos la cadena de caracteres
            # 'weather.txt' a la función mostrar_satelites.
            self.weather = Tkinter.Checkbutton(ventana_tles, text = 'Weather',\
            font = self.Verd10Roman, variable = self.variable_weather,\
            onvalue = 'weather.txt', offvalue = 0,\
            command = lambda flag = 'weather.txt': self.mostrar_satelites(flag))
            # Asignamos una posición al objeto creado.
            self.weather.grid(row = 0, column = 0, rowspan = 1,\
            columnspan = 1, sticky = Tkinter.W)

            # Generamos el objeto variable_noaa del tipo StringVar.
            self.variable_noaa = Tkinter.StringVar()
            # Creamos un Checkbutton llamado noaa. El valor que nos devolverá
            # al activarse será 'noaa.txt' devolviéndonos '0' al desactivarse.
            # Invocaremos a la función mostrar_satelites con el texto
            # 'noaa.txt' al pulsar sobre el objeto.
```

```
self.noaa = Tkinter.Checkbutton(ventana_tles, text = 'Noaa',\
font = self.Verd10Roman, variable = self.variable_noaa,\
onvalue = 'noaa.txt', offvalue = 0,\
command = lambda flag = 'noaa.txt': self.mostrar_satelites(flag))
# Asignamos una posición al objeto en la ventana.
self.noaa.grid(row = 0, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un objeto variable_goes del tipo StringVar.
self.variable_goes = Tkinter.StringVar()
# Crearemos un objeto llamado goes del tipo Checkbutton.
# Este objeto nos devolverá la cadena de caracteres 'goes.txt'
# al activar el objeto y '0' al desactivarlo.
# Al pulsarlo, en cualquiera de las ocasiones, llamaremos a la
# función mostrar_satelites con el texto 'goes.txt'.
self.goes = Tkinter.Checkbutton(ventana_tles, text = 'Goes',\
font = self.Verd10Roman, variable = self.variable_goes,\
onvalue = 'goes.txt', offvalue = 0,\
command = lambda flag = 'goes.txt': self.mostrar_satelites(flag))
# Fijamos una posición para el objeto en el LabelFrame ventana_tles.
self.goes.grid(row = 0, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un objeto del tipo StringVar llamado variable_resource.
self.variable_resource = Tkinter.StringVar()
# Generamos un nuevo objeto del tipo Checkbutton. El texto que nos
# devolverá al activarlo será 'resource.txt' siendo '0' al
# desactivarlo.
# Al hacer click en el objeto pasaremos el texto 'resource.txt' a
# la función mostrar_satelites.
self.resource = Tkinter.Checkbutton(ventana_tles,\
text = 'Resource', font = self.Verd10Roman,\
variable = self.variable_resource, onvalue = 'resource.txt',\
offvalue = 0,\
command = lambda flag = 'resource.txt':\
self.mostrar_satelites(flag))
# Asignamos una posición al objeto creado en el objeto LabelFrame.
self.resource.grid(row = 1, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un nuevo objeto del tipo StringVar llamado
# variable_sarsat.
self.variable_sarsat = Tkinter.StringVar()
# Creamos un objeto llamado sarsat del tipo Checkbutton.
# Este objeto nos devolverá el texto 'sarsat.txt' cuando se
# encuentre activo y el texto '0' al desactivarse.
# En ambos casos invocaremos a la función mostrar_satelites con el
# texto 'sarsat.txt'.
self.sarsat = Tkinter.Checkbutton(ventana_tles, text = 'Sarsat',\
font = self.Verd10Roman, variable = self.variable_sarsat,\
onvalue = 'sarsat.txt', offvalue = 0,\
command = lambda flag = 'sarsat.txt': self.mostrar_satelites(flag))
# Fijamos la posición del objeto sarsat en el LabelFrame creado.
```

```
self.sarsat.grid(row = 1, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un nuevo objeto llamado variable_dmc del tipo StringVar.
self.variable_dmc = Tkinter.StringVar()
# Generamos un nuevo objeto llamado dmc del tipo Checkbutton.
# El texto que nos devolverá será 'dmc.txt' si se encuentra en el
# estado activo y '0' si está desactivado.
# Pasaremos la cadena de texto 'dmc.txt' a la función
# mostrar_satelites en ambos casos.
self.dmc = Tkinter.Checkbutton(ventana_tles, text = 'Dmc',\
font = self.Verd10Roman, variable = self.variable_dmc,\
onvalue = 'dmc.txt', offvalue = 0,\
command = lambda flag = 'dmc.txt': self.mostrar_satelites(flag))
# Asignamos una posición al objeto creado en su LabelFrame.
self.dmc.grid(row = 1, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un nuevo objeto del tipo StringVar llamado
# variable_tdrss.
self.variable_tdrss = Tkinter.StringVar()
# Creamos el objeto tdrss del tipo Checkbutton fijando un valor
# de respuesta para el estado activado de 'tdrss.txt' y '0' para
# el estado desactivado.
# En ambos casos pasaremos la cadena de texto 'tdrss.txt' a la
# función mostrar_satelites.
self.tdrss = Tkinter.Checkbutton(ventana_tles, text = 'Tdrss',\
font = self.Verd10Roman, variable = self.variable_tdrss,\
onvalue = 'tdrss.txt', offvalue = 0,\
command = lambda flag = 'tdrss.txt': self.mostrar_satelites(flag))
# Asignamos una posición al objeto creado dentro de su LabelFrame.
self.tdrss.grid(row = 2, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos el objeto variable_geo como un objeto StringVar.
self.variable_geo = Tkinter.StringVar()
# Generamos un objeto del tipo Checkbutton con el nombre geo.
# El texto que nos devolverá será 'geo.txt' cuando esté activo y
# '0' cuando se desactive.
# Al clickar sobre este pasaremos la cadena de texto 'geo.txt' a la
# función mostrar_satelites.
self.geo = Tkinter.Checkbutton(ventana_tles, text = 'Geo',\
font = self.Verd10Roman, variable = self.variable_geo,\
onvalue = 'geo.txt', offvalue = 0,\
command = lambda flag = 'geo.txt': self.mostrar_satelites(flag))
# Fijamos la posición del objeto geo dentro del LabelFrame
# ventana_tles.
self.geo.grid(row = 2, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un nuevo objeto del tipo StringVar con el nombre
# variable_gorizont.
self.variable_gorizont = Tkinter.StringVar()
```

```
# Creamos un objeto del tipo Checkbutton llamado ventana_tles.
# Este objeto devolverá la cadena de texto 'gorizont.txt' en el
# caso de encontrarse activo y '0' cuando esté desactivado.
# Pasará el texto 'gorizont.txt' a la función mostrar_satelites
# cuando hagamos click en él.
self.gorizont = Tkinter.Checkbutton(ventana_tles,\
text = 'Gorizont', font = self.Verd10Roman,\
variable = self.variable_gorizont, onvalue = 'gorizont.txt',\
offvalue = 0,\
command = lambda flag = 'gorizont.txt':\
self.mostrar_satelites(flag))
# Asignamos una localización al objeto gorizont dentro de su
# LabelFrame.
self.gorizont.grid(row = 2, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos el objeto variable_raduga del objeto StringVar.
self.variable_raduga = Tkinter.StringVar()
# Creamos el objeto raduga del tipo Checkbutton.
# Este objeto devolverá el texto 'raduga.txt' al activarse y '0'
# al desactivarse. En ambos casos invocará a la función
# mostrar_satelites con la cadena de caracteres 'raduga.txt'.
self.raduga = Tkinter.Checkbutton(ventana_tles, text = 'Raduga',\
font = self.Verd10Roman, variable = self.variable_raduga,\
onvalue = 'raduga.txt', offvalue = 0,\
command = lambda flag = 'raduga.txt': self.mostrar_satelites(flag))
# Fijamos la posición del objeto raduga en su localización correcta.
self.raduga.grid(row = 3, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un nuevo objeto del tipo StringVar llamado
# variable_molniya.
self.variable_molniya = Tkinter.StringVar()
# Creamos un objeto llamado molniya del tipo Checkbutton.
# Este objeto devolverá el valor 'molniya.txt' cuando esté activo
# y '0' cuando esté desactivado. En ambos casos se invocará a la
# función mostrar_satelites pasándole el texto 'molniya.txt'.
self.molniya = Tkinter.Checkbutton(ventana_tles, text = 'Molniya',\
font = self.Verd10Roman, variable = self.variable_molniya,\
onvalue = 'molniya.txt', offvalue = 0,\
command = lambda flag = 'molniya.txt': self.mostrar_satelites(flag))
# Asignamos una localización al objeto molniya dentro de su
# LabelFrame.
self.molniya.grid(row = 3, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos el objeto variable_intelsat del tipo StringVar.
self.variable_intelsat = Tkinter.StringVar()
# Creamos otro objeto del tipo Checkbutton llamado intelsat.
# Este nos devolverá la cadena de caracteres 'intelsat.txt' cuando
# se encuentre activo y '0' cuando esté desactivado.
# Al activar o desactivar el botón se invocará a la función
# mostrar_satelites con el texto 'intelsat.txt'.
```

```
self.intelsat = Tkinter.Checkbutton(ventana_tles,\
text = 'Intelsat', font = self.Verd10Roman,\
variable = self.variable_intelsat, onvalue = 'intelsat.txt',\
offvalue = 0,\
command = lambda flag = 'intelsat.txt':\
self.mostrar_satelites(flag))
# Fijamos la localización del objeto intelsat dentro de su marco.
self.intelsat.grid(row = 3, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un objeto del tipo StringVar llamado variable_iridium.
self.variable_iridium = Tkinter.StringVar()
# Generamos un objeto llamado iridium del tipo Checkbutton.
# Fijaremos un valor para el estado activado de 'iridium.txt' y
# de '0' para el estado desactivado. En ambas ocasiones llamaremos
# a la función mostrar_satelites con el texto 'iridium.txt'.
self.iridium = Tkinter.Checkbutton(ventana_tles, text = 'Iridium',\
font = self.Verd10Roman, variable = self.variable_iridium,\
onvalue = 'iridium.txt', offvalue = 0,\
command = lambda flag = 'iridium.txt': self.mostrar_satelites(flag))
# Fijaremos la posición de iridium dentro del marco ventana_tles.
self.iridium.grid(row = 4, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un objeto llamado variable_orbcomm del tipo StringVar.
self.variable_orbcomm = Tkinter.StringVar()
# Creamos el objeto orbcomm del tipo Checkbutton. Este objeto
# nos devolverá el texto 'orbcomm.txt' si se encuentra activo y el
# texto '0' si no lo está. En ambos casos, al hacer click en el
# botón, el objeto llamará a la función mostrar_satelites pasándole
# la cadena de caracteres 'orbcomm.txt'.
self.orbcomm = Tkinter.Checkbutton(ventana_tles, text = 'Orbcomm',\
font = self.Verd10Roman, variable = self.variable_orbcomm,\
onvalue = 'orbcomm.txt', offvalue = 0,\
command = lambda flag = 'orbcomm.txt':\
self.mostrar_satelites(flag))
# Fijamos la posición de orbcomm dentro de su marco.
self.orbcomm.grid(row = 4, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos el objeto del tipo StringVar variable_globalstar.
self.variable_globalstar = Tkinter.StringVar()
# Generamos un nuevo objeto del tipo Checkbutton llamado globalstar.
# Este nuevo objeto nos devolverá el valor 'globalstar.txt' cuando
# se encuentre activo y '0' cuando esté desactivado.
# En ambos casos el objeto llamará a la función mostrar_valores
# pasándole la cadena de texto 'globalstar.txt'.
self.globalstar = Tkinter.Checkbutton(ventana_tles,\
text = 'Globalstar', font = self.Verd10Roman,\
variable = self.variable_globalstar, onvalue = 'globalstar.txt',\
offvalue = 0,\
command = lambda flag = 'globalstar.txt':\
self.mostrar_satelites(flag))
```

```
# Asignamos una localización a globalstar dentro de su marco.
self.globalstar.grid(row = 4, column = 2,\
rowspan = 1, columnspan = 1, sticky = Tkinter.W)

# Creamos el objeto variable_amateur del tipo StringVar.
self.variable_amateur = Tkinter.StringVar()
# Creamos el objeto amateur del tipo Checkbutton.
# Este objeto nos devolverá el valor 'amateur.txt' cuando este
# activo y '0' cuando se encuentre desactivado. Al hacer click sobre
# él se ejecutará la función mostrar_satelites con el texto
# 'amateur.txt'.
self.amateur = Tkinter.Checkbutton(ventana_tles, text = 'Amateur',\
font = self.Verd10Roman, variable = self.variable_amateur,\
onvalue = 'amateur.txt', offvalue = 0,\
command = lambda flag = 'amateur.txt':\
self.mostrar_satelites(flag))
# Fijamos la posición de amateur en el marco.
self.amateur.grid(row = 5, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un nuevo objeto del tipo StringVar con el nombre
# variable_xcomm.
self.variable_xcomm = Tkinter.StringVar()
# Generamos un objeto del tipo Checkbutton llamado xcomm.
# Este Checkbutton nos devolverá el valor 'x-comm.txt' cuando esté
# activo y '0' cuando esté desactivado. Al hacer click en el objeto
# se invocará a la función mostrar_satelites con el texto
# 'x-comm.txt'.
self.xcomm = Tkinter.Checkbutton(ventana_tles, text = 'X-Comm',\
font = self.Verd10Roman, variable = self.variable_xcomm,\
onvalue = 'x-comm.txt', offvalue = 0,\
command = lambda flag = 'x-comm.txt':\
self.mostrar_satelites(flag))
# Asignamos una posición al objeto xcomm en su marco.
self.xcomm.grid(row = 5, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un objeto llamado variable_othercomm del tipo StringVar.
self.variable_othercomm = Tkinter.StringVar()
# Creamos un objeto llamado othercomm del tipo Checkbutton.
# Asignaremos el valor 'other-comm.txt' a othercomm cuando se
# encuentre activo y '0' cuando esté desactivado.
# En ambos casos, al hacer click sobre el objeto, se llamará a la
# función mostrar_satelites pasándole el texto 'other-comm.txt'.
self.othercomm = Tkinter.Checkbutton(ventana_tles,\
text = 'Other-C.', font = self.Verd10Roman,\
variable = self.variable_othercomm, onvalue = 'other-comm.txt',\
offvalue = 0,\
command = lambda flag = 'other-comm.txt':\
self.mostrar_satelites(flag))
# Fijamos la posición de othercomm en el marco ventana_tles.
self.othercomm.grid(row = 5, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)
```

```
# Generamos un nuevo objeto del tipo StringVar llamado
# variable_gpsops.
self.variable_gpsops = Tkinter.StringVar()
# Este nuevo objeto, gpsopsn nos proporcionará la cadena de texto
# 'gps-ops.txt' cuando esté activado y '0' cuando no.
# Además, al clickar sobre este se ejecutará la función
# mostrar_satelites. Para ello le daremos el texto 'gps-ops.txt'.
self.gpsops = Tkinter.Checkbutton(ventana_tles, text = 'Gps-Ops',\
font = self.Verd10Roman, variable = self.variable_gpsops,\
onvalue = 'gps-ops.txt', offvalue = 0,\
command = lambda flag = 'gps-ops.txt':\
self.mostrar_satelites(flag))
# Asignamos una localización a gpsops en su marco.
self.gpsops.grid(row = 6, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un objeto de tipo variable_gloops llamado variable_gloops.
self.variable_gloops = Tkinter.StringVar()
# Generamos un objeto llamado gloops del tipo Checkbutton.
# Este objeto nos devolverá la cadena de texto 'glo-ops.txt' cuando
# este activado y '0' cuando no.
# Al cambiar de estado se ejecutará la función mostrar_satelites
# con la cadena de texto 'glo-ops.txt'.
self.gloops = Tkinter.Checkbutton(ventana_tles, text = 'Glo-Ops',\
font = self.Verd10Roman, variable = self.variable_gloops,\
onvalue = 'glo-ops.txt', offvalue = 0,\
command = lambda flag = 'glo-ops.txt':\
self.mostrar_satelites(flag))
# Asignamos una posición a gloops dentro del marco ventana_tles.
self.gloops.grid(row = 6, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un objeto llamado variable_galileo del tipo StringVar.
self.variable_galileo = Tkinter.StringVar()
# Generamos un nuevo objeto del tipo Checkbutton con el nombre
# galileo. Este objeto nos dará la cadena de texto 'galileo.txt'
# cuando este activado y '0' cuando no.
# Al clickar sobre el mismo se ejecutará la función
# mostrar_satelites con la cadena de caracteres 'galileo.txt'.
self.galileo = Tkinter.Checkbutton(ventana_tles, text = 'Galileo',\
font = self.Verd10Roman, variable = self.variable_galileo,\
onvalue = 'galileo.txt', offvalue = 0,\
command = lambda flag = 'galileo.txt':\
self.mostrar_satelites(flag))
# Fijamos la posición de galileo dentro de su marco.
self.galileo.grid(row = 6, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un nuevo objeto del tipo StringVar y le damos el
# nombre variable_beidou.
self.variable_beidou = Tkinter.StringVar()
# Creamos el objeto beidou del tipo Checkbutton.
```

```
# beidou nos devolverá el texto 'beidou.txt' cuando se encuentre
# activo y '0' cuando no.
# Al clickar sobre este objeto invocaremos a la función
# mostrar_satelites con la cadena de caracteres beidou.txt'.
self.beidou = Tkinter.Checkbutton(ventana_tles, text = 'Beidou',\
font = self.Verd10Roman, variable = self.variable_beidou,\
onvalue = 'beidou.txt', offvalue = 0,\
command = lambda flag = 'beidou.txt':\
self.mostrar_satelites(flag))
# Fijamos la posición de beidou.
self.beidou.grid(row = 7, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos un nuevo objeto llamado variable_sbas del tipo
# StringVar.
self.variable_sbas = Tkinter.StringVar()
# Creamos un objeto del tipo Checkbutton llamado sbas.
# Este objeto, sbas, nos devolverá la cadena de texto 'sbas.txt'
# cuando esté activo y '0' cuando esté desactivado.
# Además, al cambiar entre uno y otro estado, se ejecutará la
# función mostrar_satelites. Para ello le pasaremos la cadena
# de caracteres 'sbas.txt'.
self.sbas = Tkinter.Checkbutton(ventana_tles, text = 'Sbas',\
font = self.Verd10Roman, variable = self.variable_sbas,\
onvalue = 'sbas.txt', offvalue = 0,\
command = lambda flag = 'sbas.txt':\
self.mostrar_satelites(flag))
# Asignamos una posición al objeto sbas dentro del marco
# ventana_tles.
self.sbas.grid(row = 7, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un objeto del tipo StringVar llamado variable_nnss.
self.variable_nnss = Tkinter.StringVar()
# Generamos un nuevo objeto del tipo Checkbutton con el nombre nnss.
# Este objeto nos devolverá la cadena de texto 'nnss.txt' cuando
# se encuentra activo y '0' cuando no.
# Al clickar sobre el objeto se ejecutará la función
# mostrar_satelites pasándole el texto 'nnss.txt'.
self.nnss = Tkinter.Checkbutton(ventana_tles, text = 'Nnss',\
font = self.Verd10Roman, variable = self.variable_nnss,\
onvalue = 'nnss.txt', offvalue = 0,\
command = lambda flag = 'nnss.txt':\
self.mostrar_satelites(flag))
# Fijamos la posición del objeto nnss en su marco.
self.nnss.grid(row = 7, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos el objeto variable_musson del tipo StringVar.
self.variable_musson = Tkinter.StringVar()
# Creamos un nuevo objeto del tipo Checkbutton llamado musson.
# Este objeto nos devolverá el texto 'musson.txt' cuando se
# encuentre activo y '0' cuando no sea así.
```

```
# En ambos casos, al cambiar de estado, se ejecutará la función
# mostrar_satelites pasándole el texto 'musson.txt'.
self.musson = Tkinter.Checkbutton(ventana_tles, text = 'Musson',\
font = self.Verd10Roman, variable = self.variable_musson,\
onvalue = 'musson.txt', offvalue = 0,\
command = lambda flag = 'musson.txt':\
self.mostrar_satelites(flag))
# Asignamos una posición al objeto musson dentro del
# marco ventana_tles.
self.musson.grid(row = 8, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un objeto StringVar llamado variable_science.
self.variable_science = Tkinter.StringVar()
# Generamos un nuevo objeto del tipo Checkbutton dentro del marco
# ventana_tles.
# Este objeto nos proporcionará el texto 'science.txt' cuando este
# activado y '0' cuando no sea así. Además, al hacer click sobre él,
# se invocará la función mostrar_tles con el texto 'science.txt'.
self.science = Tkinter.Checkbutton(ventana_tles, text = 'Science',\
font = self.Verd10Roman, variable = self.variable_science,\
onvalue = 'science.txt', offvalue = 0,\
command = lambda flag = 'science.txt':\
self.mostrar_satelites(flag))
# Fijamos la posición de science dentro de su marco.
self.science.grid(row = 8, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos el objeto variable_geodetic del tipo StringVar.
self.variable_geodetic = Tkinter.StringVar()
# Creamos el objeto geodetic del tipo Checkbutton.
# Este objeto nos devolverá la cadena de texto 'geodetic.txt'
# cuando este activo y '0' cuando no. Al cambiar de estado este
# objeto invocará a la función mostrar_valores pasándole la cadena
# de texto 'geodetic.txt'.
self.geodetic = Tkinter.Checkbutton(ventana_tles,\
text = 'Geodetic', font = self.Verd10Roman,\
variable = self.variable_geodetic, onvalue = 'geodetic.txt',\
offvalue = 0,\
command = lambda flag = 'geodetic.txt':\
self.mostrar_satelites(flag))
# Asignamos la posición de geodetic en el marco ventana_tles.
self.geodetic.grid(row = 8, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos el objeto de tipo StringVar llamado variable_engineering.
self.variable_engineering = Tkinter.StringVar()
# Generamos un nuevo objeto llamado engineering de tipo Checkbutton.
# Este objeto nos devolverá el valor 'engineering.txt' cuando se
# encuentre activo y '0' cuando no.
# Al clickar sobre este ejecutaremos la función mostrar_satelites
# pasándole la cadena de texto 'engineering.txt'.
self.engineering = Tkinter.Checkbutton(ventana_tles,\
```

```
text = 'Engineering', font = self.Verd10Roman,\
variable = self.variable_engineering, onvalue = 'engineering.txt',\
offvalue = 0,\
command = lambda flag = 'engineering.txt':\
self.mostrar_satelites(flag))
# Asignamos la posición de engineering en su marco.
self.engineering.grid(row = 9, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Crearemos un objeto del tipo StringVar llamado variable_education.
self.variable_education = Tkinter.StringVar()
# Generamos un objeto llamado education del tipo Checkbutton.
# Este objeto nos dará la cadena de texto 'education.txt' cuando
# se encuentra activo y '0' cuando no.
# Al cambiar de estado llamaremos a la función mostrar_satelites
# pasándole el texto 'education.txt'.
self.education = Tkinter.Checkbutton(ventana_tles,\
text = 'Education', font = self.Verd10Roman,\
variable = self.variable_education, onvalue = 'education.txt',\
offvalue = 0,\
command = lambda flag = 'education.txt':\
self.mostrar_satelites(flag))
# Fijamos la posición de education en el marco ventana_tles.
self.education.grid(row = 9, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos el objeto variable_military del tipo StringVar.
self.variable_military = Tkinter.StringVar()
# A continuación generamos un nuevo objeto del tipo Checkbutton
# llamado military. Este objeto nos devolverá la cadena de texto
# 'military.txt' cuando se encuentra activo y '0' cuando no lo está.
# Al clickar sobre el botón llamamos a la rutina mostrar_satelites
# con el texto 'military.txt'.
self.variable_military = Tkinter.StringVar()
self.military = Tkinter.Checkbutton(ventana_tles,\
text = 'Military', font = self.Verd10Roman,\
variable = self.variable_military, onvalue = 'military.txt',\
offvalue = 0,\
command = lambda flag = 'military.txt':\
self.mostrar_satelites(flag))
# Fijamos la posición de military dentro del marco ventana_tles.
self.military.grid(row = 9, column = 2, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Creamos un objeto llamado variable_radar del tipo StringVar.
self.variable_radar = Tkinter.StringVar()
# Generamos el objeto radar del tipo Checkbutton.
# Este objeto nos devolverá la cadena de texto 'radar.txt' cuando
# este activo y '0' cuando no lo esté.
# Al cambiar de estado se ejecutará la función mostrar_satelites
# con el texto 'radar.txt'.
self.radar = Tkinter.Checkbutton(ventana_tles, text = 'Radar',\
font = self.Verd10Roman, variable = self.variable_radar,\
```

```
        onvalue = 'radar.txt', offvalue = 0,\
        command = lambda flag = 'radar.txt': self.mostrar_satelites(flag))
        # Fijamos la posición del objeto radar dentro de su marco.
        self.radar.grid(row = 10, column = 0, rowspan = 1, columnspan = 1,
                        sticky = Tkinter.W)

        # Generamos un nuevo objeto del tipo StringVar llamado
        # variable_cubesat.
        self.variable_cubesat = Tkinter.StringVar()
        # Generamos un objeto llamado cubesat del tipo Checkbutton.
        # El valor de respuesta del objeto será 'cubesat.txt' cuando esté
        # activado y '0' cuando esté desactivado.
        # Invocaremos a la función mostrar_satelites con el texto
        # 'cubesat.txt' cuando cambie de estado.
        self.cubesat = Tkinter.Checkbutton(ventana_tles, text = 'Cubesat',\
        font = self.Verd10Roman, variable = self.variable_cubesat,\
        onvalue = 'cubesat.txt', offvalue = 0,\
        command = lambda flag = 'cubesat.txt': self.mostrar_satelites(flag))
        # Asignamos una localización al objeto dentro de su marco.
        self.cubesat.grid(row = 10, column = 1, rowspan = 1,\
        columnspan = 1, sticky = Tkinter.W)

        # Creamos un nuevo objeto del tipo StringVar llamado
        # variable_other.
        self.variable_other = Tkinter.StringVar()
        # Creamos un objeto llamado other del tipo Checkbutton.
        # Este objeto nos devolverá el valor 'other.txt' cuando esté
        # activo y '0' cuando esté desactivado. En ambos casos, al hacer
        # click sobre el mismo, se ejecutará la función mostrar_satelites
        # con el texto 'other.txt'.
        self.other = Tkinter.Checkbutton(ventana_tles, text = 'Other',\
        font = self.Verd10Roman, variable = self.variable_other,\
        onvalue = 'other.txt', offvalue = 0,\
        command = lambda flag = 'other.txt': self.mostrar_satelites(flag))
        # Fijamos la posición de other dentro del marco ventana_tles.
        self.other.grid(row = 10, column = 2, rowspan = 1, columnspan = 1,
                        sticky = Tkinter.W)

    # Ejecutamos la función mostrar_tles encargada de comprobar los ficheros
    # de los que dispongo y mostrarlos activando los botones anteriores.
    self.mostrar_tles()

    # Invoco la rutina mensajes_estado para mostrar la última actualización
    # de los ficheros de elementos orbitales.
    self.mensajes_estado()

    # A continuación crearemos un objeto del tipo LabelFrame donde irán los
    # elementos necesarios para mostrar los satélites disponibles en cada
    # familia de elementos.
    ventana_satelites = Tkinter.LabelFrame(tles_pagina,\
    text = 'Satélites disponibles', width = 250, height = 290,\
    font = self.Verd12Italic)
    # Asignamos una posición a nuestro objeto dentro de la primera página
```

```
# del notebook.
ventana_satelites.grid(row = 0, column = 1, rowspan = 1,
                        columnspan = 1, sticky = Tkinter.W)
# Evitamos la propagación del marco.
ventana_satelites.grid_propagate(0)

# Creamos un objeto del tipo ScrolledList para mostrar los satélites
# disponibles en cada fichero.
# Debido a las diferentes tipografías de cada sistema operativo
# tendremos que crear el objeto de una forma distinta.
# Sistemas operativos de la familia Macintosh.
if sys.platform == "darwin":
    # Creamos el objeto llamándolo mostrar_lista_satelites.
    self.mostrar_lista_satelites = scrolledlist.ScrolledList\
    (ventana_satelites, width=27, height=10, hscroll=1)
    # Asignamos una posición al objeto en el marco.
    self.mostrar_lista_satelites.grid(row = 0, column = 0, rowspan = 1,\
    columnspan = 1, pady = 5, padx = 5)

# Sistemas operativos basados en el kernel de Linux
elif sys.platform.startswith('linux'):
    # Creamos el objeto con el nombre mostrar_lista_satelites.
    self.mostrar_lista_satelites = scrolledlist.ScrolledList\
    (ventana_satelites, width=27, height=10, hscroll=1)
    # Fijamos la posición del objeto en el marco.
    self.mostrar_lista_satelites.grid(row = 0, column = 0, rowspan = 1,\
    columnspan = 1, pady = 5, padx = 5)

# Sistemas operativos de la familia Windows
elif sys.platform == "win32":
    # Creamos el objeto asignándole el nombre mostrar_lista_satelites.
    self.mostrar_lista_satelites = scrolledlist.ScrolledList\
    (ventana_satelites, width=30, height=10)
    # Fijamos el tipo de letra que usará el objeto.
    self.mostrar_lista_satelites.listbox.configure(font = \
    self.Verd12Roman)
    # Asignamos una posición al objeto en el marco.
    self.mostrar_lista_satelites.grid(row = 0, column = 0, rowspan = 1,\
    columnspan = 1, pady = 5, padx = 5)

# Asignamos a la nueva variable var_estado el valor de la variable
# actualización creada anteriormente en la rutina mensajes_estado.
var_estado = self.actualizacion
# Creamos un objeto del tipo StringVar con el nombre text_var_estado
# para mostrar la fecha de la última actualización de los elementos
# orbitales del sistema.
text_var_estado = Tkinter.StringVar()
# El texto del objeto text_var_estado será el de la variable var_estado.
text_var_estado.set(var_estado)

# El siguiente paso será la creación de la etiqueta para mostrar la
# fecha de la actualización y los botones necesarios del marco.
# En este caso también tendremos que dividir el proceso de creación
```

```
# debido a las particularidades de cada sistema operativo
# Sistemas operativos de la familia Macintosh.
if sys.platform == "darwin":
    # Creamos una etiqueta llamada etiqueta_actualizacion para mostrar
    # la última actualización mediante la StringVar text_var_estado.
    etiqueta_actualizacion = Tkinter.Label(ventana_satelites,\
    textvariable = text_var_estado, font = self.Verd11Roman)
    # Fijamos la posición de etiqueta_actualizacion en el marco.
    etiqueta_actualizacion.grid(row = 1, column = 0, rowspan = 1,\
    columnspan = 1)

    # Crearemos un botón llamado boton_actualizar_tles con el texto
    # 'Actualización TLEs' que llamará a la función actualizar_tles.
    boton_actualizar_tles = Tkinter.Button(ventana_satelites,\
    text = 'Actualización TLEs', width = 16,\
    command = self.actualizar_tles)
    # Asignamos la posición del botón dentro del marco.
    boton_actualizar_tles.grid(row = 2, column = 0, rowspan = 1,\
    columnspan = 1, sticky = Tkinter.E)
    # Evitamos que el botón se propague dentro del marco.
    boton_actualizar_tles.grid_propagate(0)

    # Generamos un nuevo botón con el nombre boton_guardar_configuracion
    # para llamar a la rutina para guardar la configuración de los
    # elementos orbitales.
    # Esta rutina está sin preparar, actualmente solo sale de la
    # ventana.
    boton_guardar_configuracion = Tkinter.Button(tles_pagina,\
    text = 'Guardar configuración', width = 22,\
    command = ventana_configuracion.destroy)
    # Fijamos la posición del botón dentro del marco.
    boton_guardar_configuracion.grid(row = 1, column = 1, rowspan = 1,\
    columnspan = 1, sticky = Tkinter.E)
    # Impedimos el cambio de tamaño del botón dentro del marco.
    boton_guardar_configuracion.grid_propagate(0)

# Distribuciones basadas en el kernel de Linux.
elif sys.platform.startswith('linux'):
    # Creamos una etiqueta llamada etiqueta_actualizacion para mostrar
    # la última actualización mediante la StringVar text_var_estado.
    etiqueta_actualizacion = Tkinter.Label(ventana_satelites,\
    textvariable = text_var_estado, font = self.Verd11Roman)
    # Fijamos la posición de etiqueta_actualizacion en el marco.
    etiqueta_actualizacion.grid(row = 1, column = 0, rowspan = 1,\
    columnspan = 1)

    # Crearemos un botón llamado boton_actualizar_tles con el texto
    # 'Actualización TLEs' que llamará a la función actualizar_tles.
    boton_actualizar_tles = Tkinter.Button(ventana_satelites,\
    text = 'Actualización TLEs', width = 16,\
    command = self.actualizar_tles)
    # Asignamos la posición del botón dentro del marco.
    boton_actualizar_tles.grid(row = 2, column = 0, rowspan = 1,\
```

```
        columnspan = 1, sticky = Tkinter.E)
        # Evitamos que el botón se propague dentro del marco.
        boton_actualizar_tles.grid_propagate(0)

        # Generamos un nuevo botón con el nombre boton_guardar_configuracion
        # para llamar a la rutina para guardar la configuración de los
        # elementos orbitales.
        # Esta rutina está sin preparar, actualmente solo sale de la
        # ventana.
        boton_guardar_configuracion = Tkinter.Button(tles_pagina,\
        text = 'Guardar configuración', width = 22,\
        command = ventana_configuracion.destroy)
        # Fijamos la posición del botón dentro del marco.
        boton_guardar_configuracion.grid(row = 1, column = 1, rowspan = 1,\
        columnspan = 1, sticky = Tkinter.E)
        # Impedimos el cambio de tamaño del botón dentro del marco.
        boton_guardar_configuracion.grid_propagate(0)

    # Sistemas operativos de la familia Windows.
    elif sys.platform == "win32":
        # Creamos una etiqueta llamada etiqueta_actualizacion para mostrar
        # la última actualización mediante la StringVar text_var_estado.
        etiqueta_actualizacion = Tkinter.Label(ventana_satelites,\
        textvariable = text_var_estado, font = self.Verd10Roman)
        # Fijamos la posición de etiqueta_actualizacion en el marco.
        etiqueta_actualizacion.grid(row = 1, column = 0, rowspan = 1,\
        columnspan = 1)

        # Crearemos un botón llamado boton_actualizar_tles con el texto
        # 'Actualización TLEs' que llamará a la función actualizar_tles.
        boton_actualizar_tles = Tkinter.Button(ventana_satelites,\
        text = 'Actualización TLEs', width = 16,\
        command = self.actualizar_tles)
        # Asignamos la posición del botón dentro del marco.
        boton_actualizar_tles.grid(row = 2, column = 0, rowspan = 1,\
        columnspan = 1, sticky = Tkinter.E)
        # Evitamos que el botón se propague dentro del marco.
        boton_actualizar_tles.grid_propagate(0)

        # Generamos un nuevo botón con el nombre boton_guardar_configuracion
        # para llamar a la rutina para guardar la configuración de los
        # elementos orbitales.
        # Esta rutina está sin preparar, actualmente solo sale de la
        # ventana.
        boton_guardar_configuracion = Tkinter.Button(tles_pagina,\
        text = 'Guardar configuración', width = 22,\
        command = ventana_configuracion.destroy)
        # Fijamos la posición del botón dentro del marco.
        boton_guardar_configuracion.grid(row = 1, column = 1, rowspan = 1,\
        columnspan = 1, pady = 5, sticky = Tkinter.E)
        # Impedimos el cambio de tamaño del botón dentro del marco.
        boton_guardar_configuracion.grid_propagate(0)
```

```
########################################################################

# Creamos la segunda página del objeto configuracion_notebook, esta
# página se llamará conexion_pagina.
conexion_pagina = ttk.Frame(configuracion_notebook)
# Añadimos la segunda página a configuracion_notebook dándole el texto
# 'Configuracion'.
configuracion_notebook.add(conexion_pagina, text='Configuración')

# Crearemos un objeto del tipo LabelFrame para contener los controles
# necesarios para seleccionar el protocolo deseado.
ventana_protocolo_conexion = Tkinter.LabelFrame(conexion_pagina,\
text = 'Protocolo de conexión', width = 200, height = 45,\
font = self.Verd12Italic)
# Fijamos la posición del marco en la página.
ventana_protocolo_conexion.grid(row = 0, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.N)
# Evitamos la propagación del marco en la ventana.
ventana_protocolo_conexion.grid_propagate(0)

# Creamos un objeto del tipo StringVar para almacenar el protocolo
# deseado.
self.control_protocolo = Tkinter.StringVar()

# A continuación crearemos un objeto del tipo Radiobutton para
# seleccionar el protocolo deseado. Cuando pulsemos en el protocolo
# de nuestra elección se guardará una cadena de texto en el objeto
# de nuestra elección.
# Actualmente la funcionalidad deste objeto es bastante limitada debido
# a que solo tenemos un protocolo disponible.
eleccion_YS232 = Tkinter.Radiobutton(ventana_protocolo_conexion,\
text = 'YS232', font = self.Verd11Roman,\
variable = self.control_protocolo, value = 'YS232')
# Colocamos el objeto en su posición del marco.
eleccion_YS232.grid(row = 0, column = 0, rowspan = 1, columnspan = 1)

# El siguiente paso será crear otro marco en el que colocaremos
# los diferentes controles para elegir la configuración de nuestra
# conexión.
ventana_caracteristicas_conexion = Tkinter.LabelFrame(conexion_pagina,\
text = 'Características de conexión', width = 200, height = 265,\
font = self.Verd12Italic)
# Asignamos una posición a nuestro marco en la ventana.
ventana_caracteristicas_conexion.grid(row = 1, column = 0,\
rowspan = 4, columnspan = 1, sticky = Tkinter.N)
# Evitamos la propagación del marco en la ventana.
ventana_caracteristicas_conexion.grid_propagate(0)

# Crearemos la etiqueta etiqueta_puertos con el texto 'Puertos serie:'.
etiqueta_puertos = Tkinter.Label(ventana_caracteristicas_conexion,\
text = 'Puertos serie:', font = self.Verd12Roman)
# Fijamos la posición de la etiqueta creada en su marco.
etiqueta_puertos.grid(row = 0, column = 0, rowspan = 1,\
```

```
columnspan = 2, sticky = Tkinter.W)

# Crearemos un menú despegable mediante un objeto del tipo ScrolledList.
# En el mostraremos los diferentes puertos serie disponibles del
# sistema.
# En cada sistema operativo será diferente así que, otra vez, haremos
# uso de una sentencia if-elif para seleccionar el proceso correcto.
# Sistemas operativos de la serie Mac OS X.
if sys.platform == "darwin":
    # Creamos el objeto ScrolledList dandole el nombre lista_puertos.
    self.lista_puertos = scrolledlist.ScrolledList\
    (ventana_caracteristicas_conexion, width=20, height=4,\
    callback = self.seleccionar_puerto)
    # Asignamos la posición del objeto en su marco.
    self.lista_puertos.grid (row = 1, column = 0, rowspan = 1,\
    columnspan = 2, pady = 5, padx = 5)

# Distribuciones basadas en el kernel de Linux.
elif sys.platform.startswith('linux'):
    # Creamos el objeto ScrolledList dandole el nombre lista_puertos.
    self.lista_puertos = scrolledlist.ScrolledList\
    (ventana_caracteristicas_conexion, width=20, height=4,\
    callback = self.seleccionar_puerto)
    # Fijamos la posición del objeto lista_puertos en su marco.
    self.lista_puertos.grid (row = 1, column = 0, rowspan = 1,\
    columnspan = 2, pady = 5, padx = 5)

# Sistemas operativos de la familia Windows.
elif sys.platform == "win32":
    # Creamos el objeto ScrolledList dandole el nombre lista_puertos.
    self.lista_puertos = scrolledlist.ScrolledList\
    (ventana_caracteristicas_conexion, width=23, height=4,\
    callback = self.seleccionar_puerto)
    # Configuramos el tipo de fuente del objeto.
    self.lista_puertos.listbox.configure(font = self.Verd12Roman)
    # Asignamos la posición del objeto en su marco.
    self.lista_puertos.grid (row = 1, column = 0, rowspan = 1,\
    columnspan = 2, pady = 5, padx = 5)

# Ejecutamos la función listar_puertos para mostrar, en la ScrolledList
# creada anteriormente, los puertos disponibles.
self.listar_puertos()

# Crearemos un objeto del tipo StringVar llamado text_var_puertomostrar
# para mostrar el puerto seleccionado la ScrolledList anterior.
self.text_var_puertomostrar = Tkinter.StringVar()
# Asignamos un texto inicial al objeto creado.
self.text_var_puertomostrar.set('Puerto de prueba')

# Crearemos una etiqueta llamada mostrar_puerto para mostrar el puerto
# seleccionado mediante la StringVar anterior.
self.mostrar_puerto = Tkinter.Label(ventana_caracteristicas_conexion,\
textvariable = self.text_var_puertomostrar, font = self.Verd12Roman)
```

```
# Asignamos una posición a la etiqueta mostrar_puerto en su marco.
self.mostrar_puerto.grid(row = 2, column = 0, rowspan = 1,\
columnspan = 2)

# Crearemos un botón para comprobar el puerto seleccionado. Este botón
# con el texto 'Comprobar puerto' llamaá a la función comprobar_puerto
# al ser pulsado.
boton_comprobar_puerto = Tkinter.Button(\
ventana_caracteristicas_conexion, text = 'Comprobar puerto',\
font = self.Verd12Roman, width = 20, height = 1,\
command = self.comprobar_puerto)
# Asignamos una posición al botón boton_comprobar_puerto en el marco.
boton_comprobar_puerto.grid(row = 3, column = 0, rowspan = 1,\
columnspan = 2)

# Generamos un nuevo objeto del tipo etiqueta con el texto 'Velocidad:'.
etiqueta_velocidad = Tkinter.Label(ventana_caracteristicas_conexion,\
text = 'Velocidad:', font = self.Verd12Roman)
# Asignamos una posición a la etiqueta creada en el marco
# ventana_caracteristicas_conexion.
etiqueta_velocidad.grid(row = 4, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generaremos una lista llamada lista_velocidad con los diferentes
# posibles valores de la velocidad de conexión.
lista_velocidad = ['300', '1200', '2400', '4800', '9600', '14400',
                   '28800', '38400', '57600', '115200', '230400']

# Crearemos un objeto del tipo Spinbox para la selección de la velocidad
# de conexión. Los valores posibles serán los de la lista anterior.
self.menu_conexion = Tkinter.Spinbox(ventana_caracteristicas_conexion,\
font = self.Verd11Roman, values = lista_velocidad, width = 7)
# Fijaremos la posición de menu_conexion en su marco.
self.menu_conexion.grid(row = 4, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.E)

# Generaremos una nueva etiqueta llamada etiqueta_byte con el texto
# 'Longitud del byte:'.
etiqueta_byte = Tkinter.Label(ventana_caracteristicas_conexion,\
text = 'Longitud del byte:', font = self.Verd12Roman)
# Fijamos una posición a la etiqueta creada en el marco
# ventana_caracteristicas_conexion.
etiqueta_byte.grid(row = 5, column = 0, rowspan = 1, columnspan = 1,\
sticky = Tkinter.W)

# Generamos una nueva lista llamada longitud_byte con, de momento,
# un único elemento.
longitud_byte = ['8']

# Crearemos otro objeto del tipo Spinbox para mostrar las diferentes
# longitudes del byte.
self.menu_byte = Tkinter.Spinbox(ventana_caracteristicas_conexion,\
font = self.Verd11Roman, values = longitud_byte, width = 7)
```

```
# Fijamos la posición de menu_byte en el marco.
self.menu_byte.grid(row = 5, column = 1, rowspan = 1, columnspan = 1,\
sticky = Tkinter.E)

# Generamos una nueva etiqueta llamada etiqueta_paridad con el texto
# 'Paridad:'.
etiqueta_paridad = Tkinter.Label(ventana_caracteristicas_conexion,\
text = 'Paridad:', font = self.Verd12Roman)
# Asignamos la posición de etiqueta_paridad en el marco
# ventana_caracteristicas_conexion.
etiqueta_paridad.grid(row = 6, column = 0, rowspan = 1, columnspan = 1,\
sticky = Tkinter.W)

# Creamos una nueva lista llamada paridad con dos elementos 'Si' y 'No'.
paridad = ['Si', 'No']

# Creamos un nuevo ojbeto del tipo Spinbox para realizar la selección
# del valor de paridad.
self.menu_paridad = Tkinter.Spinbox(ventana_caracteristicas_conexion,\
font = self.Verd11Roman, values = paridad, width = 7)
# Asignamos una localización a menu_paridad dentro del marco
# ventana_caracteristicas_conexion.
self.menu_paridad.grid(row = 6, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.E)

# Crearemos una nueva etiqueta llamada etiqueta_bits con el texto
# 'Bits de parada:'.
etiqueta_bits = Tkinter.Label(ventana_caracteristicas_conexion,\
text = 'Bits de parada:', font = self.Verd12Roman)
# Fijamos la posición de etiqueta_bits dentro de su marco.
etiqueta_bits.grid(row = 7, column = 0, rowspan = 1,\
columnspan = 1, sticky = Tkinter.W)

# Generamos una nueva lista para almacenar los diferentes valores de
# bits de parada del sistema.
bits_parada = ['0', '1']

# Creamos el último Spinbox de este marco para seleccionar una cantidad
# de bits de parada.
self.bits_parada = Tkinter.Spinbox(ventana_caracteristicas_conexion,\
font = self.Verd11Roman, values = bits_parada, width = 7)
# Fijamos la localizacion de bits_parada dentro de su marco.
self.bits_parada.grid(row = 7, column = 1, rowspan = 1,\
columnspan = 1, sticky = Tkinter.E)

# Creamos un nuevo LabelFrame para contener controles opcionales del
# sistema.
# Aunque este se encuentra actualmente vacío está previsto añadir
# funcionalidad dentro de él.
ventana_radio = Tkinter.LabelFrame(conexion_pagina,\
text = 'Configuraciones radio', font = self.Verd12Italic, width = 330,\
height = 250 )
# Colocamos ventana_radio en la ventana.
```

```
ventana_radio.grid(row = 0, column = 1, rowspan = 3, columnspan = 1,
                   padx = 5, pady = 5, sticky = Tkinter.N)
# Evitamos la propagación de ventana_radio.
ventana_radio.grid_propagate(0)

# Generamos un nuevo objeto del tipo LabelFrame donde pondremos
# los elementos necesarios para guardar una nueva configuración de
# conexión.
ventana_guardado = Tkinter.LabelFrame(conexion_pagina,\
text = 'Guardar configuración', font = self.Verd12Italic, width = 330,\
height = 50)
# Asignamos una posición a ventana_guardado en la
# página conexion_pagina.
ventana_guardado.grid(row = 4, column = 1, rowspan = 1, columnspan = 1,\
padx = 5, pady = 5, sticky = Tkinter.N)
# Evitamos la propagación del marco dentro de la página.
ventana_guardado.grid_propagate(0)

# Dividimos la creación de los objetos del marco dependiendo del
# sistema operativo.
# Sistemas operativos de la familia Macintosh.
if sys.platform == "darwin":
    # Tamaño mínimo de la primera columna.
    ventana_guardado.columnconfigure(0, minsize = 190)
    # Tamaño mínimo de la segunda columna.
    ventana_guardado.columnconfigure(1, minsize = 90)
    # Tamaño mínimo de la tercera columna.
    ventana_guardado.columnconfigure(2, minsize = 40)

# Sistemas operativos basados en el kernel de Linux.
elif sys.platform.startswith('linux'):
    # Tamaño mínimo de la primera columna.
    ventana_guardado.columnconfigure(0, minsize = 190)
    # Tamaño mínimo de la segunda columna.
    ventana_guardado.columnconfigure(1, minsize = 90)
    # Tamaño mínimo de la tercera columna.
    ventana_guardado.columnconfigure(2, minsize = 40)

# Sistemas operativos de la familia Windows.
elif sys.platform == "win32":
    # Tamaño mínimo de la primera columna.
    ventana_guardado.columnconfigure(0, minsize = 190)
    # Tamaño mínimo de la segunda columna.
    ventana_guardado.columnconfigure(1, minsize = 90)
    # Tamaño mínimo de la tercera columna.
    ventana_guardado.columnconfigure(2, minsize = 40)

# Generamos una nueva etiqueta llamada etiqueta_nombre con el texto
# 'Nombre de la configuración:'.
etiqueta_nombre = Tkinter.Label(ventana_guardado,\
text = 'Nombre de la configuración:', font = self.Verd12Roman)
# Fijamos una localización para la etiqueta.
etiqueta_nombre.grid(row = 0, column = 0, rowspan = 1, columnspan = 1)
```

```
        # Creamos un campo de entrada para el nombre de la configuración.
        self.nombre_configuracion = Tkinter.Entry(ventana_guardado, width = 10)
        # Fijamos la posición del campo de entrada en el marco ventana_guardado.
        self.nombre_configuracion.grid(row = 0, column = 1, rowspan = 1,\
        columnspan = 1, sticky = Tkinter.E)

        # Creamos un botón para llamar a la rutina
        # guardar_configuracion_conexion.
        boton_guardar = Tkinter.Button(ventana_guardado, text = 'Ok',\
        font = self.Verd12Roman, width = 2, height = 1,\
        command = self.guardar_configuracion_conexion)
        # Asignamos una posición a boton_guardar dentro del marco
        # ventana_guardado.
        boton_guardar.grid(row = 0, column = 2, rowspan = 1,\
        columnspan = 1, sticky = Tkinter.E)

        ##########################################################################
        # Iniciamos la ventana de configuracion con la sentencia mainloop().
        ventana_configuracion.mainloop()

##############################################################################

# Rutina para mostrar que ficheros tengo o no en la libreria de elementos
# orbitales.
def mostrar_tles(self):
        # Guardamos el directorio actual de trabajo.
        directorio_trabajo = os.getcwd()
        # Nos dirigimos al directorio donde almacenamos los ficheros de
        # elementos orbitales.
        os.chdir(directorio_trabajo + '/TLEs')
        # Creamos una lista de los ficheros disponibles en la carpeta
        lista_sat = os.listdir(os.getcwd())
        # Volvemos al directorio principal.
        os.chdir(directorio_trabajo)
        # Eliminamos de la lista que creamos anteriormente el elemento
        # flag.txt, este no nos interesa puesto que solo es para almacenar
        # la fecha de la última actualización de los ficheros.
        lista_sat.pop(lista_sat.index("flag.txt"))

        # A continuación comprobaremos la disponibilidad de los ficheros de
        # elementos orbitales. Para ello los iremos buscando uno a uno
        # en la lista creada anteriormente.
        # Si están en la lista activaremos el objeto de tipo Checkbutton
        # correspondiente
        # Si no están en la lista es que no se encuentran disponibles en
        # nuestra librería y tendremos que desactivar el objeto de tipo
        # Checkbutton asociado.
        # Librería noaa.
        try:
                lista_sat.index("noaa.txt")
                self.noaa.select()
        except ValueError:
```

```
        self.noaa.deselect()

    # Libreria weather.
    try:
        lista_sat.index("weather.txt")
        self.weather.select()
    except ValueError:
        self.weather.deselect()

    # Libreria goes.
    try:
        lista_sat.index("goes.txt")
        self.goes.select()
    except ValueError:
        self.goes.deselect()

    # Libreria resource.
    try:
        lista_sat.index("resource.txt")
        self.resource.select()
    except ValueError:
        self.resource.deselect()

    # Libreria sarsat.
    try:
        lista_sat.index("sarsat.txt")
        self.sarsat.select()
    except ValueError:
        self.sarsat.deselect()

    # Libreria dmc.
    try:
        lista_sat.index("dmc.txt")
        self.dmc.select()
    except ValueError:
        self.dmc.deselect()

    # Liberia tdrss.
    try:
        lista_sat.index("tdrss.txt")
        self.tdrss.select()
    except ValueError:
        self.tdrss.deselect()

    # Libreria geo.
    try:
        lista_sat.index("geo.txt")
        self.geo.select()
    except ValueError:
        self.geo.deselect()

    # Libreria gorizont.
    try:
```

```
        lista_sat.index("gorizont.txt")
        self.gorizont.select()
    except ValueError:
        self.gorizont.deselect()

    # Libreria raduga.
    try:
        lista_sat.index("raduga.txt")
        self.raduga.select()
    except ValueError:
        self.raduga.deselect()

    # Libreria molniya.
    try:
        lista_sat.index("molniya.txt")
        self.molniya.select()
    except ValueError:
        self.molniya.deselect()

    # Libreria intelsat.
    try:
        lista_sat.index("intelsat.txt")
        self.intelsat.select()
    except ValueError:
        self.intelsat.deselect()

    # Libreria iridium.
    try:
        lista_sat.index("iridium.txt")
        self.iridium.select()
    except ValueError:
        self.iridium.deselect()

    # Libreria intelsat.
    try:
        lista_sat.index("intelsat.txt")
        self.intelsat.select()
    except ValueError:
        self.intelsat.deselect()

    # Libreria orbcomm.
    try:
        lista_sat.index("orbcomm.txt")
        self.orbcomm.select()
    except ValueError:
        self.orbcomm.deselect()

    # Libreria globalstar.
    try:
        lista_sat.index("globalstar.txt")
        self.globalstar.select()
    except ValueError:
        self.globalstar.deselect()
```

```
# Libreria amateur.
try:
    lista_sat.index("amateur.txt")
    self.amateur.select()
except ValueError:
    self.amateur.deselect()

# Libreria x-comm.
try:
    lista_sat.index("x-comm.txt")
    self.xcomm.select()
except ValueError:
    self.xcomm.deselect()

# Libreria other-comm.
try:
    lista_sat.index("other-comm.txt")
    self.othercomm.select()
except ValueError:
    self.othercomm.deselect()

# Libreria gps-ops.
try:
    lista_sat.index("gps-ops.txt")
    self.gpsops.select()
except ValueError:
    self.gpsops.deselect()

# Libreria glo-ops.
try:
    lista_sat.index("glo-ops.txt")
    self.gloops.select()
except ValueError:
    self.gloops.deselect()

# Libreria galileo.
try:
    lista_sat.index("galileo.txt")
    self.galileo.select()
except ValueError:
    self.galileo.deselect()

# Libreria beidou.
try:
    lista_sat.index("beidou.txt")
    self.beidou.select()
except ValueError:
    self.beidou.deselect()

# Libreria sbas.
try:
    lista_sat.index("sbas.txt")
```

```
        self.sbas.select()
    except ValueError:
        self.sbas.deselect()

    # Libreria nnss.
    try:
        lista_sat.index("nnss.txt")
        self.nnss.select()
    except ValueError:
        self.nnss.deselect()

    # Libreria musson.
    try:
        lista_sat.index("musson.txt")
        self.musson.select()
    except ValueError:
        self.musson.deselect()

    # Libreria science.
    try:
        lista_sat.index("science.txt")
        self.science.select()
    except ValueError:
        self.science.deselect()

    # Libreria geodetic.
    try:
        lista_sat.index("geodetic.txt")
        self.geodetic.select()
    except ValueError:
        self.geodetic.deselect()

    # Libreria engineering.
    try:
        lista_sat.index("engineering.txt")
        self.engineering.select()
    except ValueError:
        self.engineering.deselect()

    # Libreria education.
    try:
        lista_sat.index("education.txt")
        self.education.select()
    except ValueError:
        self.education.deselect()

    # Libreria military.
    try:
        lista_sat.index("military.txt")
        self.military.select()
    except ValueError:
        self.military.deselect()
```

```
        # Libreria radar.
        try:
            lista_sat.index("radar.txt")
            self.radar.select()
        except ValueError:
            self.radar.deselect()

        # Libreria cubesat.
        try:
            lista_sat.index("cubesat.txt")
            self.cubesat.select()
        except ValueError:
            self.cubesat.deselect()

        # Libreria other.
        try:
            lista_sat.index("other.txt")
            self.other.select()
        except ValueError:
            self.other.deselect()

    # La siguiente función crea un objeto de la clase ActualizarTLEs y muestra
    # por pantalla el mensaje de estado correspondiente indicando la fecha
    # de la última actualización.
    def mensajes_estado(self):
        # Importamos el fichero con la clase ActualizarTLEs
        import funciones_suplementarias
        # Creamos un objeto llamado actualización de la clase ActualizarTLEs.
        # Asignamos a los dos atributos necesarios para crear la clase,
        # flag_actualizar y lista_actualización, los valores 0 y nulo.
        # Eso quiere decir que no actualizará los ficheros y que pasará un valor
        # vacio como lista.
        actualizacion = funciones_suplementarias.ActualizarTLEs(0,None)

        # Asignamos al objeto actualización (de la clase StringVar) el valor
        # del atributo salida del objeto actualizacion.
        self.actualizacion = actualizacion.salida

    # Rutina para actualizar los ficheros de elementos orbitales de nuestra
    # libreria.
    def actualizar_tles(self):
        # Creamos una lista vacia.
        lista_tipos_satelites = []

        # Comprobamos el estado de todos los checkbutton y lo anotamos
        # en la lista lista_tipos_satelites.
        lista_tipos_satelites.append(self.variable_weather.get())
        lista_tipos_satelites.append(self.variable_noaa.get())
        lista_tipos_satelites.append(self.variable_resource.get())
        lista_tipos_satelites.append(self.variable_sarsat.get())
        lista_tipos_satelites.append(self.variable_dmc.get())
        lista_tipos_satelites.append(self.variable_tdrss.get())
        lista_tipos_satelites.append(self.variable_geo.get())
```

```
        lista_tipos_satelites.append(self.variable_gorizont.get())
        lista_tipos_satelites.append(self.variable_raduga.get())
        lista_tipos_satelites.append(self.variable_molniya.get())
        lista_tipos_satelites.append(self.variable_intelsat.get())
        lista_tipos_satelites.append(self.variable_iridium.get())
        lista_tipos_satelites.append(self.variable_orbcomm.get())
        lista_tipos_satelites.append(self.variable_globalstar.get())
        lista_tipos_satelites.append(self.variable_amateur.get())
        lista_tipos_satelites.append(self.variable_xcomm.get())
        lista_tipos_satelites.append(self.variable_othercomm.get())
        lista_tipos_satelites.append(self.variable_gpsops.get())
        lista_tipos_satelites.append(self.variable_gloops.get())
        lista_tipos_satelites.append(self.variable_galileo.get())
        lista_tipos_satelites.append(self.variable_beidou.get())
        lista_tipos_satelites.append(self.variable_sbas.get())
        lista_tipos_satelites.append(self.variable_nnss.get())
        lista_tipos_satelites.append(self.variable_musson.get())
        lista_tipos_satelites.append(self.variable_science.get())
        lista_tipos_satelites.append(self.variable_geodetic.get())
        lista_tipos_satelites.append(self.variable_education.get())
        lista_tipos_satelites.append(self.variable_engineering.get())
        lista_tipos_satelites.append(self.variable_military.get())
        lista_tipos_satelites.append(self.variable_radar.get())
        lista_tipos_satelites.append(self.variable_cubesat.get())
        lista_tipos_satelites.append(self.variable_other.get())


        # Creamos un objeto de la clase "ActualizarTLEs" para actualizar los tles
        # Importamos el fichero funciones_suplementarias.
        import funciones_suplementarias
        # Generamos un objeto llamado actualizacion del tipo ActualizarTLEs.
        # Asignamos a los dos atributos necesarios para crear la clase,
        # flag_actualizar y lista_actualización, los valores 1 y
        # lista_tipo_satelites.
        # Eso quiere decir que actualizaremos los ficheros y que cambiaremos
        # los ficheros presentes en lista_tipo_satelites.
        actualizacion = funciones_suplementarias.ActualizarTLEs(1,\
        lista_tipos_satelites)

        # Ejecuto la rutina mensajes_estado para mostrar la fecha de la última
        # actualización de los ficheros.
        self.mensajes_estado()

    # La función mostrar_satelites nos permitirá mostrar en el ScrolledList
    # mostrar_lista_satelites los satélites disponible en el ficheros
    # seleccionado. Para ello tendremos que pasarle como argumento el nombre
    # de este.
    def mostrar_satelites(self, flag):
        # Guardamos en la variable directorio_trabajo el directorio actual.
        directorio_trabajo = os.getcwd()
        # Nos dirigimos a la carpeta donde almacenamos los elementos orbitales.
        os.chdir(directorio_trabajo + '/TLEs')
```

```
# Encapsularemos todas las acciones en una sentencia try-except.
# Así, si el fichero seleccionado no existe, no se interrumpirá la
# ejecución del programa.
try:
    # Abrimos el archivo con el nombre flag y almacenamos su contenido
    # en listado_satelites_tle
    listado_satelites_tle = open(flag, 'r')
    # Leemos línea por línea el contenido almacenado.
    listado_nombres_satelites = listado_satelites_tle.readlines()
    # Guardamos las líneas como elementos independientes de una lista.
    listado_nombres_satelites = \
    [item.rstrip('\n') for item in listado_nombres_satelites]
    # Regresamos al directorio original del programa.
    os.chdir(directorio_trabajo)

    # A continuación creamos una lista con los divisores de tres del
    # numero total de líneas. Esto será necesario para extraer los
    # nombre de los satélites de la lista ya que estos se encuentran
    # cada tres líneas del fichero.
    # Importamos el fichero funciones_suplementarias.
    import funciones_suplementarias
    # Creamos un objeto llamado lista del tipo Multiplos3.
    # Le pasamos como argumento la longitud total de la lista
    # listado_nombres_satelites.
    lista = funciones_suplementarias.Multiplos3\
    (len(listado_nombres_satelites))

    # Asociamos a la nueva variable al atributo numeros_lista del objeto
    # creado anteriormente. Este atributo es una lista con los divisores
    # del número total de líneas del fichero analizado entre tres.
    a = lista.numeros_lista

    # Borramos los valores que hubiera presentes en el ScrolledList
    # mostrar_lista_satelites.
    self.mostrar_lista_satelites.clear()

    # Agregamos los nombres a mostrar_lista_satelites.
    # Para ello creamos un bucle for que recorrerá todos los valores
    # desde 0 hasta la longitud total de la lista a, la lista con los
    # divisores de 3 del total de líneas.
    for i in range(len(a)):
        # La función append añade un elemento nuevo al objeto sin
        # modificar los anteriores.
        self.mostrar_lista_satelites.append\
        (listado_nombres_satelites[a[i]])

# En el caso de no existir el fichero el programa saltará a esta línea.
except IOError:
    # Mostraremos un aviso por línea de comandos.
    print "Archivo no disponible"
    # Volvemos al directorio de trabajo del script.
    os.chdir(directorio_trabajo)
```

```
# Rutina para seleccionar el puerto de la lista de puertos disponibles.
def seleccionar_puerto(self, indice_puerto):
    # Creamos el atributo de la clase self.indice_puerto a partir del valor
    # de indice_puerto.
    self.indice_puerto = indice_puerto

# Rutina para listar los puertos serie disponibles en el sistema.
# Para ello utilizaremos el módulo pyserial. Si necesitamos más información
# del funcionamiento de este tendremos que acudir a su sitio web oficial.
# Este variará dependiendo del sistema operativo que utilicemos.
def listar_puertos(self):
    # Creo una lista vacia para almacenar los puertos disponibles.
    puertos_serie = []
    # Mac OS X.
    if sys.platform == 'darwin':
        # Importamos las funciones necesarias del módulo.
        from serial.tools import list_ports_osx
        # Creamos un bucle for para recorrer los puertos disponibles.
        for port in list_ports_osx.comports():
            # En cada iteración del bucle añadiremos a la lista
            # puertos_serie un nuevo nombre de puerto.
            puertos_serie.append(port[0])
        # Con la lista creada, mediante otro bucle for, añadiremos los
        # valores de esta al ScrolledList lista_puertos.
        for i in range(len(puertos_serie)):
            self.lista_puertos.append(puertos_serie[i])

    # Linux.
    elif sys.platform.startswith('linux'):
        # Almacenamos los puertos disponibles cuya dirección empieza por
        # /dev/tty
        puertos_serie = glob.glob('/dev/tty*')
        # Añadiremos los puertos de la lista al ScrolledList lista_puertos.
        for i in range(len(puertos_serie)):
            self.lista_puertos.append(puertos_serie[i])

    # Windows.
    elif sys.platform == "win32":
        # Importamos el módulo necesario.
        import serial.tools.list_ports
        # Guardamos los puertos serie disponibles en la lista puertos_serie.
        puertos_serie = serial.tools.list_ports_windows.comports()
        # Mediante un bucle for añadimos los puertos disponibles al
        # ScrolledList lista_puertos.
        for i in range(len(puertos_serie)):
            self.lista_puertos.append(puertos_serie[i])

# Con la rutina guardar_configuracion_conexion añadiremos una nueva
# configuración de conexión a nuestro listado de configuraciones.
# Para ello recogeremos los valores seleccionados en los widgets anteriores
# y los copiaremos en una nueva línea de texto en el fichero
# confconexion.list
def guardar_configuracion_conexion(self):
```

```
    # Encapsulamos las acciones en un fichero para así evitar que el
    # funcionamiento del programa se detenga si falta algún dato.
    try:
        # Utilizando el móduclo csv abrimos el fichero confconexion.list
        # en modo "añadir líneas".
        datos_conexion = csv.writer(open("confconexion.list", 'ab'))
        # Escribimos una nueva línea con los siguientes datos:
        # nombre de la configuración, velocidad de conexion, longitud del
        # byte, paridad, cantidad de bits de parada y dirección del puerto.
        datos_conexion.writerow([self.nombre_configuracion.get(),\
        self.menu_conexion.get(), self.menu_byte.get(),\
        self.menu_paridad.get(), self.bits_parada.get(),\
        self.lista_puertos[self.indice_puerto]])
    # Si falta algún dato o el fichero no se puede abrir saltará a esta
    # línea
    except:
        # Mostraremos por línea de comandos un aviso.
        print "Faltan datos de conexión"

################################################################################
################################################################################

# Funciones del sistema necesarias para el funcionamiento del mismo.

# Rutina para listar las ciudades del archivo CSV "localizaciones.list".
# Al ejecutarse se podrán ver por pantalla las localizaciones disponibles
# en nuestra base de datos.
def ciudades (self):
    # Abre el archivo tipo .CSV con las caracteristicas de las ciudades.
    listar_ciudades = csv.reader(open("localizaciones.list", 'rb'))
    # Lee la primera columna del archivo
    for row in listar_ciudades:
        # Añade las nombres de las ciudades a lista_ciudades, que es un
        # objeto del tipo ScrolledList.
        self.lista_ciudades.append(row[0])


# Rutina para mostrar las características de la ciudad seleccionada. Esta
# función se ejecutará cuando pulsemos sobre una localización del
# objeto lista_ciudades.
# Para ello tendremos que pasarle el indice del listado donde se encuentra
# la localización seleccionada.
def seleccionar_ciudad(self, indice_ciudad):
    # Abrimos el fichero localizaciones.list en modo lectura.
    listar_observador = csv.reader(open("localizaciones.list", 'rb'))
    # Extraemos la línea del fichero de localizaciones.
    self.ciudad = indice_ciudad

    # Para llegar a la localización que deseemos utilizaremos un bucle for
    # que recorrerá todas las líneas hasta llegar a la elegida.
    for i in range(indice_ciudad + 1):
        # observador es la variable donde se almacena el valor de
        # cada línea. En cada iteración su contenido varia.
```

```
        observador = listar_observador.next()
        # Formateamos los elementos de la lista para mostrarlos en pantalla.
        # Mostramos el nombre de la localización, que es el primer elemento
        # de la lista.
        self.text_var_localizacionbuena.set(observador[0])
        # Mostramos la latitud separando los valores de grados, minutos y
        # segundos por dos puntos.
        self.text_var_latitudbuena.set("%s:%s:%s" % (observador[1],\
        observador[2], observador[3]))
        # Mostramos el valor de la longitud de la localización separando
        # también grados, minutos y segundos por dos puntos.
        self.text_var_longitudbuena.set("%s:%s:%s" %(observador[4],\
        observador[5], observador[6]))
        # Mostramos la altitud de la localización como un único elemento.
        self.text_var_altitudbuena.set("%s" %(observador[7]))
        # Por limitaciones de espacio en pantalla el horizonte no se podrá
        # visualizar en los sistemas operativos de la familia Windows.
        # Sistemas operativos de la familia Macintosh.
        if sys.platform == "darwin":
            self.text_var_horizontebueno.set("%s" %(observador[9]))
        # Distribuciones basadas en el kernel de Linux.
        elif sys.platform.startswith('linux'):
            self.text_var_horizontebueno.set("%s" %(observador[9]))
        # En cualquier otro caso no hacemos nada.
        else:
            pass

        # Formateamos los datos para pasarselos a la rutina de cálculo.
        # Nombre de la localización.
        self.observador_actual = observador[0]
        # Latitud del observador.
        self.latitud_actual = "%s:%s:%s" % (observador[1], observador[2],
                                            observador[3])
        # Longitud del observador.
        self.longitud_actual = "%s:%s:%s" % (observador[4], observador[5],
                                             observador[6])
        # Altitud de la localización.
        self.altitud_actual ="%s" %(observador[7])
        # Presión de la ubicación del observador.
        self.presion_actual = "%s" %(observador[8])
        # Horizonte medio del observador.
        self.horizonte_actual ="%s" %(observador[9])

    # Llamamos a la función familias. Está función nos mostrará las familias
    # de satélites disponibles.
    self.familias()

    # Para acomodar el mensaje de aviso a la pantalla pondremos un texto
    # diferente dependiendo del sistema operativo.
    # Texto para los sistemas operativos de la familia Macintosh.
    if sys.platform == 'darwin':
        self.text_var_eventobueno.set('Seleccione familia del objeto')
    # Texto para Linux.
```

```
        elif sys.platform.startswith('linux'):
            self.text_var_eventobueno.set('Seleccione familia del objeto')
        # Texto para Windows.
        elif sys.platform == "win32":
            self.text_var_eventobueno.set('Seleccione familia')

    # Rutina para listar las familias de elementos orbitales disponibles.
    def familias(self):
        # Guardamos el directorio actual de trabajo.
        directorio_trabajo = os.getcwd()
        # Nos dirigimos al directorio donde se encuentran los elementos.
        os.chdir(directorio_trabajo + '/TLEs')
        # Creamos una lista con los archivos disponibles en la carpeta.
        self.listado_TLEs = os.listdir(os.getcwd())
        # Eliminamos el elemento 'flag.txt' de la lista listado_TLEs.
        self.listado_TLEs.pop(self.listado_TLEs.index("flag.txt"))
        # Añadimos los elementos de la anterior lista al ScrolledList
        # lista_tipossatelite.
        for i in range(len(self.listado_TLEs)):
            # Recortamos las líneas a una longitud de 18 carácteres para
            # que se vean correctamente por pantalla.
            self.lista_tipossatelite.append(self.listado_TLEs[i][:18])
        # Regresamos al directorio original de trabajo.
        os.chdir(directorio_trabajo)

    # Esta función, seleccionar_familia, se ejecutará cuando hagamos click
    # sobre un elemento del ScrolledList lista_tipossatelite.
    # Tendremos que pasarle el indice de la línea sobre la que pulsemos.
    def seleccionar_familia(self, indice_familia):

        # Guardamos el nombre de la familia elegida en la variable TLE_actual.
        self.TLE_actual = self.listado_TLEs[indice_familia]
        # Limpiamos los elementos del ScrolledList lista_satelites.
        self.lista_satelites.clear()
        # Ejecutamos la función satelites. Esta función listará los satélites
        # de la familia seleccionada en el ScrolledList lista_satelites.
        self.satelites()
        # Mostramos un aviso por pantalla pidiendo al usuario que seleccione
        # un satélite del listado.
        self.text_var_eventobueno.set('Seleccione satélite')

    # Con la rutina satelites mostraremos los nombres de los satélites
    # pertenecientes a la familia que escojamos.
    # Para ello tendremos que abrir el archivo correspondiente y leer las líneas
    # del mismo.
    def satelites(self):
        # Almacenamos el directorio de trabajo del programa.
        directorio_trabajo = os.getcwd()
        # Nos dirigmos a la carpeta donde se almacena los elementos orbitales.
        os.chdir(directorio_trabajo + '/TLEs')
        # Abrimos en modo lectura el archivo correspondiente a la familia
        # elegida.
        listado_satelites_tle = open(self.TLE_actual, 'r')
```

```
        # Leemos todas las líneas del fichero.
        listado_nombres_satelites = listado_satelites_tle.readlines()
        # Creamos una lista en la que cada línea es un elemento de esta.
        listado_nombres_satelites = \
        [item.rstrip('\n') for item in listado_nombres_satelites]
        # Regresamos al directorio inicial de trabajo.
        os.chdir(directorio_trabajo)

        # Importamos el fichero funciones_suplementarias.
        import funciones_suplementarias
        # Crearemos un objeto llamado lista del tipo Multiplos3.
        # Le damos el valor de la longitud de listado_nombres_satelites.
        lista = funciones_suplementarias.Multiplos3\
        (len(listado_nombres_satelites))

        # Crearemos la variable a y la igualaremos al atributo numeros_lista del
        # objeto lista anteriormente creado.
        # Este atributo es una lista con los divisores del número total de
        # líneas del fichero analizado entre tres.
        a = lista.numeros_lista

        # Agrega las líneas cuyo numero esté en la lista a. Estas serán
        # los nombres de los satélites.
        for i in range(len(a)):
            self.lista_satelites.append(listado_nombres_satelites[a[i]][:18])

    # Cuando hagamos click sobre un satélite del lista se invocará a esta
    # rutina. Para ejecutarla tendremos que pasarle el índice del listado.
    def seleccionar_satelite(self, indice_satelite):

        # Igualaremos self.satelite actual al nombre del satélite elegido.
        self.satelite_actual = self.lista_satelites[indice_satelite]
        # Mostraremos el nombre del objeto seleccionado por pantalla.
        self.text_var_satelitebueno.set(self.satelite_actual)

        # Convertimos la variable indice_satelite, una variable de esta función
        # en una variable de la clase para poder acceder a ella desde cualquier
        # parte de la misma.
        self.satelite = indice_satelite

        # Debido a cuestiones de espacio fijaremos un mensaje de aviso distinto
        # dependiendo del sistema operativo.
        # Sistemas operativos de la familia Macintosh.
        if sys.platform == 'darwin':
            self.text_var_eventobueno.set('Fije modo de seguimiento')
        # Distribuciones basadas en el kernel de Linux.
        elif sys.platform.startswith('linux'):
            self.text_var_eventobueno.set('Fije modo de seguimiento')
        # Sistemas operativos de la familia Windows.
        elif sys.platform == "win32":
            self.text_var_eventobueno.set('Fije seguimiento')

        # Activamos el botón de seguimiento para permitir al usuario que inicie
```

```
        # las rutinas de seguimiento del sistema.
        self.boton_seguimiento.config(state = Tkinter.ACTIVE)
        # También habilitamos el botón de posición para que pueda acceder
        # a la posición actual del objeto.
        self.boton_posicion.config(state = Tkinter.ACTIVE)

        # Llamamos a la rutina fijar_modo_actualizacion para que el usuario
        # pueda elegir como se comportara el sistema, es decir, cada cuanto
        # actualizará los valores que ofrece.
        self.fijar_modo_actualizacion()

    # La función fijar_valores_conexion leerá del fichero confconexion.list
    # las características de la configuración deseada y los cargará como
    # variables de la clase.
    def fijar_valores_conexion(self):

        # Abrimos el fichero confconexion.list con el módulo CSV y lo cargamos
        # en la variable listar_configuraciones_conexiones.
        listar_configuraciones_conexiones = csv.reader\
        (open("confconexion.list", 'rb'))

        # Obtenemos el índice de la configuración deseada.
        indice_configuracion = self.lista_configuraciones.index\
        (self.spinbox_configuraciones.get())

        # Mostramos un aviso por pantalla con el nombre de la configuración
        # seleccionada.
        self.text_var_mostrarestado.set('Configuración %s fijada' % \
        (self.spinbox_configuraciones.get()))

        # Recorremos las diferentes configuraciones posibles hasta dar con la
        # nuestra.
        # Cargaremos los valores necesarios.
        for i in range(indice_configuracion + 1):
            # configuracion es la variable donde se almacena el valor de
            # cada línea. En cada iteración su contenido varia.
            configuracion = listar_configuraciones_conexiones.next()
            # Asigno el valor de cada elemento de la lista a una variable
            # diferente.
            # El primer elemento será el nombre de la conexión.
            self.nombre_conexion = configuracion[0]
            # El segundo será la velocidad de conexión del sistema.
            self.velocidad_conexion = configuracion[1]
            # El tercero elemento nos indicará el número de bits de cada byte.
            self.bytes_conexion = configuracion[2]
            # La paridad o no de la conexión vendrá indicada en el cuarto
            # elemento.
            self.paridad_conexion = configuracion[3]
            # Los bits de parada se indicarán en el quinto elemento.
            self.parada_conexion = configuracion[4]
            # La dirección del puerto se encontrará en el último elemento.
            self.puerto_conexion = configuracion [5]
```

```
# Función no implementada. En un futuro comprobará que el puerto elegido
# está operativo.
def comprobar_puerto(self):
    print "Compruebo puerto"

# La rutina establecer_conexion comprueba la conexión del sistema
# preguntando al rotor la posición actual del rotor.
def establecer_conexion(self):
    # Encapsulamos las acciones de la rutina en una sentencia try-except
    # para evitar que el programa se cuelgue si surge algún error.
    try:
        # Importamos el fichero protocolo_serie.
        import protocolo_serie
        # Creamos un objeto llamado conexión del tipo YS232 iniciándolo
        # con los parámetros de configuración fijados anteriormente.
        conexion = protocolo_serie.YS232(self.nombre_conexion,\
        self.velocidad_conexion, self.bytes_conexion,\
        self.paridad_conexion, self.parada_conexion, self.puerto_conexion)
        # Invocamos a la función comprobar_conexion del objeto conexion.
        conexion.comprobar_conexion()
        # Mostramos el mensaje de aviso estado del objeto conexion.
        self.text_var_mostrarestado.set(conexion.estado)
    # Si algo falla el programa saltará a este punto.
    except:
        # Mostraemos un mensaje de aviso en la pantalla.
        self.text_var_mostrarestado.set("Fallo en la conexión")

# Rutina para fijar los valores por defecto guardados en un archivo de
# configuración.
# Esta rutina aún no está implementada de momento solo muestra por línea de
# comandos la hora de inicio del programa.
def valores_por_defecto(self):

    # Creamos un objeto llamado tiempo de la clase time del modulo time.
    tiempo = time.time()
    # Creamos un nuevo objeto llamado a de la clase gmtime heredado del
    # objeto tiempo.
    a = time.gmtime(tiempo)
    # Creamos un nuevo objeto llamado c de la clase localtime heredado del
    # objeto tiempo.
    c = time.localtime(tiempo)

    # Igualamos la variable self.minuto al atributo tm_min del objeto a.
    self.minuto = a.tm_min
    # Igualamos la variable self.segundo al atributo tm_sec del objeto a.
    self.segundo = a.tm_sec
    # Igualamos la variable self.horalocal al atributo tm_hour del objeto c.
    self.horalocal = c.tm_hour

    # Creamos una variable tiempolocal mostrando las tres variables
    # generadas anteriormente.
    tiempolocal = "%s:%s:%s" %(self.horalocal, self.minuto, self.segundo)
```

```
    # Mostramos por la línea de comandos el inicio del programa.
    print "Inicio del programa a las %s" %(tiempolocal)

# Con la rutina fijar_modo_actualizacion fijaremos el modo de actualización
# del sistema. Este podrá ser automático o manual, esto es, asignando un
# tiempo a la actualización de las coordenadas y de la posición de los
# rotores.
def fijar_modo_actualizacion(self):
    # Encapsulamos las rutinas dentro de una sentencia try-except para
    # evitar cuelgues del programa.
    try:
        # Si hemos seleccionado 'Si' en el widget control_actualizacion
        # saltaremos a este punto.
        if self.control_actualizacion.get() == 'Si':
            # Deshabilitaremos el campo de entrada del tiempo de
            # actualización de la visualización de las coordenadas.
            self.posibles_actualizaciones_coordenadas.config(state = \
            Tkinter.DISABLED)
            # Deshabilitaremos el campo de entrada del tiempo de
            # actualización del envío de coordenadas a los rotores.
            self.posibles_actualizaciones_rotores.config(state = \
            Tkinter.DISABLED)

            # Fijaremos un tiempo de actualización de la coordenadas en
            # pantalla de un segundo. Un tiempo inferior no tendrá sentido.
            self.tiempo = 1000
            # Asignamos un tiempo de actualización mínimo a las coordenadas
            # que se envían a los rotores de medio segundo.
            self.tiempo_actualizacion = 500

            # Activaremos el botón de seguimiento.
            self.boton_seguimiento.config(state = Tkinter.ACTIVE)

        # Si hemos seleccionado 'No' se activarán los campos para la
        # introducción de los diferentes tiempos.
        elif self.control_actualizacion.get() == 'No':
            # Activamos el campo para fijar la actualización de la
            # visualización de las coordenadas.
            self.posibles_actualizaciones_coordenadas.config(state = \
            Tkinter.NORMAL)
            # Habilitamos el campo para asignar un tiempo de actualización
            # mínimo a las coordenadas que envíaremos a los rotores.
            self.posibles_actualizaciones_rotores.config(state = \
            Tkinter.NORMAL)

            # Llamo a la rutina para fijar los tiempos.
            self.fijar_actualizaciones()

            # Activaremos el botón de seguimiento.
            self.boton_seguimiento.config(state = Tkinter.ACTIVE)

        # En el caso de que no hayamos seleccionado ni 'Si' ni 'No'
        # saltaremos a este punto.
```

```
            # Mostraremos un mensaje de aviso para el usuario recordándole que
            # tiene que elegir un modo de seguimiento.
            else:
                # Sistemas operativos de la familia Macintosh.
                if sys.platform == 'darwin':
                    self.text_var_eventobueno.set('Fije modo de seguimiento')
                # Distribuciones basadas en el kernel de Linux.
                elif sys.platform.startswith('linux'):
                    self.text_var_eventobueno.set('Fije modo de seguimiento')
                # Sistemas operativos de la familia Windows.
                elif sys.platform == "win32":
                    self.text_var_eventobueno.set('Fije seguimiento')

        # Si algo falla saltamos a este punto.
        except:
            # Mostraremos un mensaje de aviso por línea de comandos.
            print "Error en el modo de actualizacion"

    # La siguiente rutina, fijar_actualizaciones, asigna los tiempos que el
    # usuario ha elegido a la ejecución del programa.
    def fijar_actualizaciones(self):
        # Para evitar que el programa continue su ejecución sin un tiempo
        # definido meteremos todas las rutinas en una sentencia try-except.
        # Así, mientras no haya un tiempo, el programa mostrará por pantalla
        # el mensaje de advertencia "Introduzca un intervalo".
        try:
            # Obtenemos el valor del campo posibles_actualizaciones_coordenadas.
            tiempo = self.posibles_actualizaciones_coordenadas.get()
            # Convertimos la variable tiempo de tipo string a un tipo entero.
            tiempo = int(tiempo)
            # Ya que el tiempo se fija en milisegundos multiplicamos por 1000
            # para poder trabajar con segundos.
            self.tiempo = tiempo*1000

            # Obtenemos el valor del campo posibles_actualizaciones_rotores.
            tiempo_actualizacion = self.posibles_actualizaciones_rotores.get()
            # Convertimos la variable tiempo_actualizacion de un tipo string
            # a un tipo entero.
            tiempo_actualizacion = int(tiempo_actualizacion)
            # Multiplicamos tiempo_actualización por 1000 para obtener su valor
            # en milisegundos.
            self.tiempo_actualizacion = tiempo_actualizacion*1000

        # Si aún no hemos introducido ningún valor nos iremos a este punto
        # del código.
        except:
            # Mostramos por pantalla el aviso "Introduzca un intervalo".
            self.text_var_eventobueno.set('Introduzca un intervalo')

    # Crearemos una función llamada tack que se encargará de iniciar el
    # seguimiento del sistema. Esta función creará dos hilos. Uno de ellos
    # encargado de mostrar por pantalla las coordenadas y el otro de enviar las
    # coordenadas a los motores.
```

```
def tack(self):

    # Mostramos por pantalla un avisado recordando que el seguimiento se
    # ha activado.
    self.text_var_eventobueno.set('Seguimiento activado')
    # Deshabilitamos el botón para activar el seguimiento.
    self.boton_seguimiento.config(state = Tkinter.DISABLED)
    # Desactivamos el botón para mostrar la posición actual del objeto.
    self.boton_posicion.config(state = Tkinter.DISABLED)
    # Invocamos a la función para fijar los tiempos de actualizaciones.
    self.fijar_actualizaciones

    # Creamos un hilo llamado coordenadas para la función
    # loop_mostrar_coordenadas.
    coordenadas = threading.Thread(target = self.loop_mostrar_coordenadas,\
    name='Mostrar coordenadas')

    # Iniciamos el hilo coordenadas.
    coordenadas.start()

    # Generamos un hilo nuevo llamado rotores para la función
    # loop_enviar_coordenadas.
    rotores = threading.Thread(target = self.loop_enviar_coordenadas,\
    name='Mover rotores')

    # Iniciamos el hilo rotores.
    rotores.start()

# La función loop_mostrar_coordenadas creará un bucle que se encargará
# de mostrar las coordenadas según los tiempo fijados anteriormente.
def loop_mostrar_coordenadas(self):
    # Llamamos a la rutina crear_coordenadas para obtener las
    # coordenadas del objeto en un momento dado.
    self.crear_coordenadas()

    # Mostramos las coordenadas por pantalla mediante la función
    # mostrar coordenadas()
    self.mostrar_coordenadas()

    # Intentamos iniciar el bucle. Encapsulamos la sentencia en un
    # try-except.
    try:
        # Mediante la función after el script ejecutará
        # loop_mostrar_coordenadas cada vez que pase tiempo.
        self.seguimiento_id = root.after(self.tiempo,\
        self.loop_mostrar_coordenadas)

    # Si falla algo el programa saltará a este punto.
    except:
        # Mostraremos por línea de comandos un aviso.
        print "Error en el proceso de mostrar las coordenadas"

# La rutina loop_enviar_coordenadas creará un bucle que se encargará
```

```
# de crear las coordenadas para el rotor según los tiempos fijados.
def loop_enviar_coordenadas(self):
    # La función desplazamiento_rotor_id creará un bucle que repetirá
    # la rutina loop_enviar_coordenadas cada vez que el tiempo fijado
    # por tiempo_actualización pase.
    self.desplazamiento_rotor_id = root.after(self.tiempo_actualizacion,\
    self.loop_enviar_coordenadas)

    # Importamos el fichero protocolo_serie.
    import protocolo_serie

    # Llamamos a la funcion crear_coordenadas para obtener las coordenadas.
    self.crear_coordenadas()

    # Preparamos las coordenadas mediante preparar_coordenadas.
    self.preparar_coordenadas()

    # Declaramos dos listas vacías llamadas lista_acimut y lista_altura.
    self.lista_acimut = []
    self.lista_altura = []

    # El sistema solo enviará la orden de movimiento si ha cambiado
    # más de la precision fijada.
    self.precision_acimut = 0.5
    self.precision_altura = 0.5

    # Igualemos la variable acimut a la variable acimut_grad. c
    acimut = int(self.acimut_grad)
    # Igualamos la variable altura a la variable altura_grad.
    altura = int(self.altura_grad)

    # Convertimos acimut en una variable de tipo string.
    acimut = str(acimut)

    # Crearemos una rutina para comprobar si el acimut ha cambiado para ello
    # iremos almacenando los valores del acimut en un fichero temporal
    # y compararemos el último con el penúltimo elemento del mismo.
    # Abrimos el fichero lista_temporal_acimut.
    archivo_temporal_acimut = open('lista_temporal_acimut.txt', 'a')
    # Escribimos el valor del acimut y un salto de línea.
    archivo_temporal_acimut.write("%s\n"% (acimut))
    # Cerramos el fichero.
    archivo_temporal_acimut.close()

    # Volvemos a abrir el fichero anterior esta vez en modo de lectura.
    lectura_temporal_acimut = open('lista_temporal_acimut.txt', 'r')
    # Leemos todas las líneas del fichero y las guardamos en lista_acimut.
    self.lista_acimut = lectura_temporal_acimut.readlines()
    # Convertimos lista_acimut en una lista donde cada línea del fichero
    # es un elemento de la lista.
    self.lista_acimut = [item.rstrip('\n') for item in self.lista_acimut]
    # Cerramos el fichero.
    lectura_temporal_acimut.close()
```

```
# A continuación compararemos el último y el penúltimo valor de la
# lista. En el caso de no tener los valores necesarios saltaremos
# directamente a la sentencia except.
try:
    # Compararemos el último valor con el penúltimo.
    if float(self.lista_acimut[-1]) > (float(self.lista_acimut[-2]) +\
    self.precision_acimut):
        # Encapsulamos las sentencias en un try-except. Si hay algún
        # problema con el envio de la posición saltaremos a la
        # sentencia except.
        try:
            # Crearemos un objeto llamado conexión de la clase YS232.
            # Le pasaremos los valores de configuración deseados.
            conexion = protocolo_serie.YS232(self.nombre_conexion,\
            self.velocidad_conexion, self.bytes_conexion,\
            self.paridad_conexion, self.parada_conexion,\
            self.puerto_conexion)
            # Enviamos la posición ejecutando la función
            # enviar_posicion del objeto conexion.
            conexion.enviar_posicion(self.acimut_grad,\
            self.altura_grad)
            # Mostramos un aviso por pantalla.
            self.text_var_mostrarestado.set('Moviendo rotor en acimut')

        # Mostramos un aviso.
        except:
            # Texto de aviso para los sistemas operativos Macintosh.
            if sys.platform == 'darwin':
                self.text_var_eventobueno('Error en el envío de la\
                 posición')
            # Texto de aviso para las distribuciones basadas en el
            # kernel de Linux.
            elif sys.platform.startswith('linux'):
                self.text_var_eventobueno('Error en el envío de la\
                 posición')
            # Texto de aviso para los sistemas operativos Windows.
            elif sys.platform == "win32":
                self.text_var_eventobueno('Error en la posición')

    # Si los valores son iguales muestra un aviso por pantalla
    # para recordar que el sistema sigue parado.
    else:
        self.text_var_mostrarestado.set('Rotor detenido')

# Si saltamos a esta sentencia damos por hecho que el sistema acaba de
# iniciarse y, por lo tanto, tenemos que enviar las ordenes inciales
# de movimiento.
except:
    # Creamos un objeto llamado conexión del tipo YS232 con los
    # parámetros de la conexión correctos.
    conexion = protocolo_serie.YS232(self.nombre_conexion,\
    self.velocidad_conexion, self.bytes_conexion,\
```

```
        self.paridad_conexion, self.parada_conexion, self.puerto_conexion)
        # Enviamos el acimut y la altura requeridas.
        conexion.enviar_posicion(self.acimut_grad, self.altura_grad)

    # Rutina para comprobar si la altura ha cambiado
    altura = str(altura)

    # Crearemos una rutina para comprobar si la altura ha cambiado para ello
    # iremos almacenando los valores de la altura en un fichero temporal
    # y compararemos el último con el penúltimo elemento del mismo.
    # Abrimos el fichero lista_temporal_altura.
    archivo_temporal_altura = open('lista_temporal_altura.txt', 'a')
    # Escribimos el valor de la altura y un salto de línea.
    archivo_temporal_altura.write("%s\n"% (altura))
    # Cerramos el fichero.
    archivo_temporal_altura.close()

    # Volvemos a abrir el fichero anterior esta vez en modo de lectura.
    lectura_temporal_altura = open('lista_temporal_altura.txt', 'r')
    # Leemos todas las líneas del fichero y las guardamos en lista_altura.
    self.lista_altura = lectura_temporal_altura.readlines()
    # Convertimos lista_altura en una lista donde cada línea del fichero
    # es un elemento de la lista.
    self.lista_altura = [item.rstrip('\n') for item in self.lista_altura]
    # Cerramos el fichero.
    lectura_temporal_altura.close()

    # A continuación compararemos el último y el penúltimo valor de la
    # lista. En el caso de no tener los valores necesarios saltaremos
    # directamente a la sentencia except.
    try:
        # Compararemos el último valor con el penúltimo.
        if float(self.lista_altura[-1]) > (float(self.lista_altura[-2]) +
                                           self.precision_altura):
            # Encapsulamos las sentencias en un try-except. Si hay algún
            # problema con el envio de la posición saltaremos a la
            # sentencia except.
            try:
                # Crearemos un objeto llamado conexión de la clase YS232.
                # Le pasaremos los valores de configuración deseados.
                conexion = protocolo_serie.YS232(self.nombre_conexion,\
                self.velocidad_conexion, self.bytes_conexion,\
                self.paridad_conexion, self.parada_conexion,\
                self.puerto_conexion)
                # Enviamos la posición ejecutando la función
                # enviar_posicion del objeto conexion.
                conexion.enviar_posicion(self.acimut_grad,\
                self.altura_grad)
                # Mostramos un aviso por pantalla.
                self.text_var_mostrarestado.set('Moviendo rotor en\
                elevación')

            # Mostramos un aviso por pantalla.
```

```
                except:
                    # Texto de aviso para los sistemas operativos Macintosh.
                    if sys.platform == 'darwin':
                        self.text_var_eventobueno('Error en el envío de la\
                         posición')
                    # Texto de aviso para las distribuciones basadas en el
                    # kernel de Linux.
                    elif sys.platform.startswith('linux'):
                        self.text_var_eventobueno('Error en el envío de la\
                         posición')
                    # Texto de aviso para los sistemas operativos Windows.
                    elif sys.platform == "win32":
                        self.text_var_eventobueno('Error en la posición')

            # Si los valores son los mismos y no hay cambios avisaremos con
            # un mensaje por pantalla.
            else:
                self.text_var_mostrarestado.set('Rotor detenido')

        # Si saltamos a esta sentencia damos por hecho que el sistema acaba de
        # iniciarse y, por lo tanto, tenemos que enviar las ordenes inciales
        # de movimiento.
        except:
            # Creamos un objeto llamado conexión del tipo YS232 con los
            # parámetros de la conexión correctos.
            conexion = protocolo_serie.YS232(self.nombre_conexion,\
            self.velocidad_conexion, self.bytes_conexion,\
            self.paridad_conexion, self.parada_conexion, self.puerto_conexion)
            # Ejecutamos la función enviar_posición del objeto conexion para
            # enviar el acimut y la altura requeridos.
            conexion.enviar_posicion(self.acimut_grad, self.altura_grad)

    # Rutina para mostrar la hora y las coordenadas solo una vez. Esta función
    # no realizará ninguna conexión con los motores.
    def mostrar_unavez_datos(self):
        # Invocamos a la función crear_coordenadas para tener los valores
        # necesarios.
        self.crear_coordenadas()

        # Ejecutamos la función mostrar_coordenadas para mostrar las coordenadas
        # creadas por pantalla.
        self.mostrar_coordenadas()

        # A continuación realizaremos la comprobación de la altura del objeto.
        # si la altura es inferior a 0 no mostraremos las coordenadas.
        if self.alturaennumeros < 0:
            # Mostraremos un mensaje de aviso.
            # Mensaje de aviso en sistemas operativos de la familia Macintosh.
            if sys.platform == 'darwin':
                self.text_var_eventobueno.set('Objeto debajo del horizonte')

            # Mensaje de aviso para distribuciones basadas en el kernel
            # de Linux.
```

```
            elif sys.platform.startswith('linux'):
                self.text_var_eventobueno.set('Objeto debajo del horizonte')

            # Mensaje de aviso para los sistemas operativos de la familia
            # Windows.
            elif sys.platform == "win32":
                self.text_var_eventobueno.set('Debajo del horizonte')

            # Intentaremos cancelar ambos bucles, si no están activos no haremos
            # nada.
            # Cancelamos el bucle encargado de mostrar las coordenadas.
            try:
                root.after_cancel(self.seguimiento_id)
            # Si no está activo no hacemos nada.
            except:
                pass

            # Cancelamos el bucle encargado de enviar las coordenadas
            # a los motores.
            try:
                root.after_cancel(self.desplazamiento_rotor_id)
            # Si no está activo no hacemos nada.
            except:
                pass

        # En el caso de ser superior, el objeto está sobre el horizonte,
        # mostraremos el aviso "Datos actuales".
        elif self.alturaennumeros > 0:
            self.text_var_eventobueno.set('Datos actuales')

    # La rutina mostrar_hora mostrará la hora en pantalla.
    def mostrar_hora(self):
        # Nombramos el proceso.
        logging.debug('Hora lanzada')

        # Crearemos un bucle llamado hora_id que ejecutará cada 1000
        # milisegundos la función mostrar_hora.
        self.hora_id = root.after(1000, self.mostrar_hora)

        # Creamos un objeto llamado tiempo de la clase time del modulo time.
        tiempo = time.time()
        # Creamos un nuevo objeto llamado a de la clase gmtime heredado del
        # objeto tiempo.
        a = time.gmtime(tiempo)
        # Creamos un nuevo objeto llamado c de la clase localtime heredado del
        # objeto tiempo.
        c = time.localtime(tiempo)

        # Igualamos la variable self.dia al atributo tm_mday del objeto a.
        self.dia = a.tm_mday
        # Igualamos la variable self.mes al atributo tm_mon del objeto a.
        self.mes = a.tm_mon
        # Igualamos la variable self.anio al atributo tm_year del objeto a.
```

```
        self.anio = a.tm_year
        # Igualamos la variable self.hora al atributo tm_hour del objeto a.
        self.hora = a.tm_hour
        # Igualamos la variable self.minuto al atributo tm_min del objeto a.
        self.minuto = a.tm_min
        # Igualamos la variable self.segundo al atributo tm_sec del objeto a.
        self.segundo = a.tm_sec

        # Igualamos la variable self.horalocal al atributo tm_hour del objeto c.
        self.horalocal = c.tm_hour

        # Generamos una nueva variable tiempo con las tres variables self.hora,
        # self.minuto y self.segundo.
        tiempo = "%s:%s:%s" %(self.hora, self.minuto, self.segundo)
        # Creamos una variable tiempolocal mostrando las tres variables
        # generadas anteriormente.
        tiempolocal = "%s:%s:%s" %(self.horalocal, self.minuto,\
        self.segundo)
        # Creamos una variable fecha con los atributos de la clase dia, mes y
        # anio.
        fecha = "%s.%s.%s" %(self.dia, self.mes, self.anio)

        # Modificamos los valores de las etiquetas del tiempo y de la fecha.
        # Hora universal.
        self.text_var_horabuena.set(tiempo)
        # Hora local.
        self.text_var_horalocalbuena.set(tiempolocal)
        # Fecha local.
        self.text_var_fechabuena.set(fecha)

    # Creamos una función llamada crear_coordenadas destinada a crear las
    # coordenadas del satélite escogido.
    def crear_coordenadas(self):

        # Importamos el fichero con la rutina de calculo.
        import calculo
        # Creamos un objeto llamado objeto_calculo de la clase Calculo.
        # Le pasaremos los valores de la localización del observador.
        objeto_calculo = calculo.Calculo(None, None, None, None, None, None,\
        None, self.observador_actual, self.longitud_actual,\
        self.latitud_actual, self.altitud_actual, self.horizonte_actual,\
        self.presion_actual, self.satelite, self.TLE_actual, None, None,\
        None, None)

        # Obtengo las coordenadas actuales del objeto deseado.
        # Igualo la variable self.altura al atributo altura del objeto
        # objeto_calculo.
        self.altura = objeto_calculo.altura
        # Igualo la variable self.acimut al atributo acimut del objeto
        # objeto_calculo.
        self.acimut = objeto_calculo.acimut
        # Igualo la variable self.ascencionrecta al atributo ascensionrecta
        # del objeto objeto_calculo.
```

```
        self.ascensionrecta = objeto_calculo.ascensionrecta
        # Igualo la variable self.declinacion al atributo declinacion
        # del objeto objeto_calculo.
        self.declinacion = objeto_calculo.declinacion
        # Igualo la variable self.orto al atributo orto del objeto
        # objeto_calculo.
        self.orto = objeto_calculo.orto
        # Convierto al atributo altura de la clase Ventana_principal en una
        # variable de tipo flotante y lo igual a la variable
        # self.alturaennumeros.
        self.alturaennumeros = float(self.altura)

    # Generamos una rutina que modificará los objetos de tipo StringVar de la
    # ventana para mostrar las coordenadas del objeto.
    def mostrar_coordenadas(self):

        # Primero comprobamos que el objeto se encuentra encima del horizonte.
        if self.alturaennumeros > 0:
            # Forzaremos la conversión de los valores a cadena de caracteres
            # para así mostrarlos por pantalla.
            altura = str(self.altura)
            acimut = str(self.acimut)
            ascensionrecta = str(self.ascensionrecta)
            declinacion = str(self.declinacion)
            orto = str(self.orto)

            # Modificamos los valores de las etiquetas de coordenadas.
            self.text_var_alturabuena.set(altura)
            self.text_var_acimutbueno.set(acimut)
            self.text_var_ascensionbuena.set(ascensionrecta)
            self.text_var_declinacionbuena.set(declinacion)
            self.text_var_ortobueno.set(orto)

        # En el caso de que se encuentre por debajo del horizonte intentaremos
        # cancelar la actualización de las coordenadas.
        else:
            # Cancelación de la actualización de las coordenadas en pantalla.
            try:
                root.after_cancel(self.seguimiento_id)
            except:
                pass

            # Cancelación de la actualización de las coordenadas para los
            # motores.
            try:
                root.after_cancel(self.desplazamiento_rotor_id)
            except:
                pass

            # Mensaje de aviso para los sistemas operativos de la familia
            # Macintosh.
            if sys.platform == 'darwin':
                self.text_var_eventobueno.set('Objeto debajo del horizonte')
```

```
            # Mensaje de aviso para las distribuciones basadas en el kernel
            # de Linux.
            elif sys.platform.startswith('linux'):
                self.text_var_eventobueno.set('Objeto debajo del horizonte')

            # Mensaje de aviso para los sistemas operativos de la familia
            # Windows.
            elif sys.platform == "win32":
                self.text_var_eventobueno.set('Debajo del horizonte')

            # Ponemos en blanco el valor de la altura del objeto.
            self.text_var_alturabuena.set('')
            # Ponemos en blanco el valor del acimut del objeto.
            self.text_var_acimutbueno.set('')
            # Ponemos en blanco el valor de la ascensión recta del objeto.
            self.text_var_ascensionbuena.set('')
            # Ponemos en blanco el valor de la declinación del objeto.
            self.text_var_declinacionbuena.set('')
            # Ponemos en blanco el valor del próximo orto del objeto.
            self.text_var_ortobueno.set('')

    # La función preparar_coordenadas redondea el número de grados de acimut
    # y altura y los prepara para el rotor. Para ello usa una clase del módulo
    # funciones_suplementarias.
    def preparar_coordenadas(self):

        # Importamos el fichero funciones_suplementarias.
        import funciones_suplementarias

        # Creamos el objeto objeto_coordenadas de la clase PrepararCoordenadas
        # pasándole los valores de acimut, altura y altura en números flotantes.
        objeto_coordenadas = funciones_suplementarias.PrepararCoordenadas\
        (self.acimut, self.altura, self.alturaennumeros)

        # Igualamos los atributos acimut_grad y altura_grad del objeto
        # objeto_coordenadas a self.acimut_grad y altura_grad.
        self.acimut_grad = objeto_coordenadas.acimut_grad
        self.altura_grad = objeto_coordenadas.altura_grad


    # La rutina parada parará las funciones del sistema.
    def parada(self):
        # Hacemos sonar el altavoz del sistema.
        root.bell(displayof = 0)
        # Intentamos detener el bucle seguimiento_id encargado de mostrar
        # las coordenadas de los objetos por pantalla.
        try:
            root.after_cancel(self.seguimiento_id)

        # Si no podemos saltamos aquí.
        except:
            # No hacemos nada, damos por hecho que el bucle no se había
```

```
            # iniciado anteriormente.
            pass

        # Intentamos detener el bucle desplazamiento_rotor_id encargado del
        # movimiento de los rotores.
        try:
            root.after_cancel(self.desplazamiento_rotor_id)

        # Si no podemos saltamos a este punto.
        except:
            # Tampoco hacemos nada ya que damos por hecho que este bucle
            # no se había iniciado con anterioridad.
            pass

        # Deshabilitamos los botones boton_seguimiento y boton_posicion
        # para forzar al usuario a introducir los datos necesarios de nuevo
        # antes de poder pulsarlos otra vez.
        self.boton_seguimiento.config(state = Tkinter.DISABLED)
        self.boton_posicion.config(state = Tkinter.DISABLED)

        # Importamos el fichero protocolo_serie.
        import protocolo_serie
        # Creamos un objeto llamado conexion de la clase YS232.
        conexion = protocolo_serie.YS232(self.nombre_conexion,\
        self.velocidad_conexion, self.bytes_conexion, self.paridad_conexion,\
        self.parada_conexion, self.puerto_conexion)
        # Invocamos al método parar_sistema del objeto conexion.
        conexion.parar_sistema()
        # Mostramos un aviso por pantalla.
        self.text_var_eventobueno.set('Sistema detenido')

        # A continuación abriremos para escritura los dos archivos temporales
        # de acimut y altura y los cerraremos de nuevo. Así borraremos su
        # contenido.
        # Primero empezaremos con el archivo destinado a los datos temporales
        # de acimut.
        archivo_temporal_acimut = open('lista_temporal_acimut.txt', 'w')
        # Para evitar problemas cerraremos el archivo.
        archivo_temporal_acimut.close()

        # El segundo paso será abrir el fichero de alturas temporales.
        archivo_temporal_altura = open('lista_temporal_altura.txt', 'w')
        # Como en el caso anterior también tendremos que cerrar el archivo.
        archivo_temporal_altura.close()

    # La función salida es la rutina para salir del sistema.
    # En esta función crearemos una ventana anidada con los controles necesarios
    # para destruir el programa.
    # También nos ocuparemos de borrar los archivos temporales generados por
    # el código del script.
    def salida(self):
        # Hacemos sonar la campana del sistema.
        root.bell(displayof = 0)
```

```
# Creamos el objeto para la ventana anidada y le damos
# el nombre ventana_salida.
ventana_salida = Tkinter.Toplevel()
# Le damos un tamaño de 155 píxeles de ancho y 96 píxeles de alto.
# Dejamos un margen interior de 5 píxeles.
ventana_salida.geometry("155x96+5+5")
# Llamamos a la ventana 'Salida'.
ventana_salida.title("Salida")
# Configuramos un margen exterior de 5 píxeles.
ventana_salida.config(padx = 5, pady = 5)

# Creamos un marco llamado salida en la ventana recién creada.
salida = Tkinter.LabelFrame(ventana_salida, width = 145, height = 86)
# Colocamos el marco en la ventana.
salida.grid(row = 0, column = 0, rowspan = 1, columnspan = 1)
# Cancelado la propagación del marco por la ventana.
salida.grid_propagate(0)

# Fijamos un tamaño mínimo de 42 píxeles a la primera fila.
salida.rowconfigure(0, minsize = 42)
# Fijamos un tamaño mínimo de 42 píxeles a la segunda fila.
salida.rowconfigure(1, minsize = 42)
# Asignamos un tamaño mínimo de 145 píxeles a la primera columna.
salida.columnconfigure(0, minsize = 145)

# A continuación abriremos para escritura los dos archivos temporales
# de acimut y altura y los cerraremos de nuevo. Así borraremos su
# contenido.
# Primero empezaremos con el archivo destinado a los datos temporales
# de acimut.
archivo_temporal_acimut = open('lista_temporal_acimut.txt', 'w')
# Para evitar problemas cerraremos el archivo.
archivo_temporal_acimut.close()

# El segundo paso será abrir el fichero de alturas temporales.
archivo_temporal_altura = open('lista_temporal_altura.txt', 'w')
# Como en el caso anterior también tendremos que cerrar el archivo.
archivo_temporal_altura.close()

# Crearemos una etiqueta con el texto '¿Desea salir del sistema?'.
# Debido a problemas de espacio este texto estará repartido en dos
# filas.
etiqueta_salida = Tkinter.Label(salida, font = self.Verd12Roman,\
text ="¿Desea salir \ndel sistema?", width = 12, height = 2,\
justify = Tkinter.CENTER)
# Colocaremos la etiqueta en el centro de la ventana.
etiqueta_salida.grid(row = 0, column = 0, rowspan = 1, columnspan = 1)
# Fijaremos el tamaño del marco salida impidiendo su propagación.
etiqueta_salida.grid_propagate(0)

# Crearemos un botón para llamar a la rutina destroy.
# Esta rutina, al estar vinculada a la ventana principal root, cerrará
```

```
        # nuestro programa.
        boton_salida = Tkinter.Button(salida, font = self.Verd12Roman,\
        text = 'Salir', width = 10, command = root.destroy)
        # Colocamos el botón debajo de la etiqueta que generamos anteriormente.
        boton_salida.grid(row = 1, column = 0, rowspan = 1, columnspan = 1)
        # Evitamos el cambio de tamaño del botón.
        boton_salida.grid_propagate(0)

################################################################################
################################################################################

if __name__ == '__main__':
    # Creamos el objeto root de la clase Tk perteneciente al módulo Tkinter.
    root = Tkinter.Tk();
    # Llamamos a la ventana creada 'Polux 0.1'.
    root.title('Polux 0.1');
    # Fijamos a la ventana unt amaño de 600 píxeles de ancho por 373 píxeles
    # de alto. Le damos un margen interior de 5 píxeles.
    root.geometry("600x373+5+5")
    # Asignamos a la primera columna un tamaño mínimo de 200 píxeles.
    root.columnconfigure(0, minsize = 200)
    # Asignamos a la segunda columna un tamaño mínimo de 200 píxeles.
    root.columnconfigure(1, minsize = 200)
    # Asignamos a la tercera columna un tamaño mínimo de 200 píxeles.
    root.columnconfigure(2, minsize = 200)
    # Creamos un objeto de la clase Ventana_principal.
    widget = Ventana_principal()
    # Iniciamos la ventana principal.
    root.mainloop()
```

Rutina de cálculo.

Rutina de cálculo de las coordenadas contenida en el fichero calculo.py.

```
# Este archivo usa el encoding: utf-8

# Clase encargada de obtener los valores de calculo y devolver los valores de
# posicion del observador y los valores del objeto seleccionado


class Calculo():

	# Función constructora de la clase. A esta función le damos los valores
	# del observador, del tiempo y del objeto elegido.
	def __init__ (self, timetuple = None, dia = None, mes = None, anio =
None,\
	hora = None, minuto = None, segundo = None, observador = None,\
	longitud = None, latitud = None, elevacion = None, horizonte = None,\
	presion = None, satelite = None, tle = None, altura = None, acimut =
None,\
	ascensionrecta = None, declinacion = None):

		# Importamos el módulo time encargado de gestionar el tiempo.
		import time
		# Cargamos el módulo ephem encargado de calcular las efemérides de
		# los objetos.
		import ephem
		# Cargamos el módulo os destinado a las tareas del sistema.
		import os

		# Creamos un objeto llamado tiempo de la clase time del modulo
		# time.
		tiempo = time.time()
		# Creamos un nuevo objeto llamado a de la clase gmtime heredado del
		# objeto tiempo.
		a = time.gmtime(tiempo)
		# Creamos una tupla de valores llamada b con los valores del año
		# del sistema, el mes, el día, la hora, el minuto y el segundo.
		b = a.tm_year, a.tm_mon, a.tm_mday, a.tm_hour, a.tm_min, a.tm_sec
		# Asignamos la tupla al atributo, de la clase Calculo, timetuple.
		self.timetuple = b

		# Asignamos el valor del día al atributo de la clase llamado dia.
		self.dia = a.tm_mday
		# Asignamos el valor del mes al atributo de la clase llamado mes.
		self.mes = a.tm_mon
		# Asignamos el valor del año al atributo de la clase llamado anio.
		self.anio = a.tm_year
		# Asignamos el valor de la hora al atributo de la clase llamado
		# hora.
		self.hora = a.tm_hour
		# Asignamos el valor del minuto al atributo de la clase llamado
```

```
# minuto.
self.minuto = a.tm_min
# Asignamos el valor del segundo al atributo de la clase
# llamado segundo.
self.segundo = a.tm_sec


# Cambio la variable satelite de una cadena de caracteres a un
# entero.
satelite = int(satelite)
# Multiplo por tres el índice y sumo uno y dos para así obtener la
# localización de las líneas necesarias en el fichero.
titulo_TLE = satelite*3
indice_linea1 = titulo_TLE + 1
indice_linea2 = titulo_TLE + 2

# Transformamos la variable elevacion de un tipo string a un tipo
# float.
elevacion = float(elevacion)
# Transformamos la variable presion de un tipo string a un tipo
# float.
presion = float(presion)
# Transformamos la variable horizonte de una cadena de caracteres
# a un tipo flotante.
horizonte = float(horizonte)

# Creamos un objeto de la clase Observer llamado observador.
observador = ephem.Observer()
# Asignamos la longitud del observador al atributo lon del objeto
# observador.
observador.lon = longitud
# Igualamos la latitud de la localización al atributo lat de
# observador.
observador.lat = latitud
# Asignamos la elevación media del observador al atributo elevation
# del objeto observador.
observador.elevation = elevacion
# Fijamos el valor de la presión media de la localización igualando
# este al atributo pressure del objeto observador.
observador.pressure = presion
# Asignamos el horizonte medio del observador al atributo horizon
# del objeto observador.
observador.horizon = horizonte
# Cargamos el tiempo actual del sistema como un atributo del
# objeto.
observador.date = ephem.Date(self.timetuple)

# Guardamos el directorio actual de trabajo.
directorio_trabajo = os.getcwd()
# Nos dirigimos al directorio de elementos orbitales.
os.chdir(directorio_trabajo + '/TLEs')

# Abrimos el archivo de elementos orbitales correspondiente.
```

```
lectura_TLEs = open(tle, 'r')
# Leemos todas las líneas del fichero y las almacenamos en la
# variable lista_TLEs.
lista_TLEs = lectura_TLEs.readlines()
# Convertimos las líneas del archivo anterior en elementos de una
# lista.
lista_TLEs = [item.rstrip('\n') for item in lista_TLEs]
# A la variable objeto le asignamos el texto "satelite".
# No es relevante para el funcionamiento.
objeto = "satelite"
# Asignamos a la variable linea1 el valor del elemento
# correspondiente de la lista anterior.
linea1 = lista_TLEs[indice_linea1]
# Fijamos en la variable linea2 el valor del elemento
# correspondiente de la lista lista_TLEs.
linea2 = lista_TLEs[indice_linea2]
# Creamos un objeto de la clase readtle con los valores anteriores.
v = ephem.readtle(objeto, linea1, linea2)
# Ejecutamos el método compute del objeto anterior asignándole los
# valores del observador.
v.compute(observador)

# Intenamos mostrar el valor del próximo orto del sistema.
try:
       orto_time = observador.next_pass(v)[0]
# Si el objeto no es visible nunca desde la localización del
# observador mostraremos un mensaje de error.
except ValueError:
       orto_time = "Siempre debajo del horizonte"

# Regresamos al directorio original de trabajo.
os.chdir(directorio_trabajo)

# Asignamos los valores obtenidos del calculo a los atributos de
# nuestra clase.
self.altura = v.alt
self.acimut = v.az
self.ascensionrecta = v.ra
self.declinacion = v.dec
self.orto = orto_time
```

Protocolo de comunicación.

Protocolo de comunicación entre el sistema y la interfaz de conexión con los rotores contenida en el fichero protocolo_serie.py.

```
# Este archivo usa el encoding: utf-8

# Modulo con clases para obtener los valores de acimut y altura deseados
# y enviarlos al rotor del modo deseado.
# De momento solo disponemos del protocolo YS232.

# Importamos el módulo encargado de las comunicaciones.
import serial


# La primera clase de este fichero será YS232. Esta clase, como su propio nombre
# indica, es la encargada de las comunicaciones mediante el protocolo YS232B.
class YS232():

	# Rutina constructora de la clase. En ella recogeremos los valores de
	# configuración de la conexión y los pasaremos al resto de funciones
	# de la clase.
	def __init__(self, nombre, baudrate, bytelength, parity, stopbits, port):

		# Convertimos los parámetros del constructor a parámetros
		# propios de la clase.
		# Nombre de la conexión.
		self.nombre = nombre
		# Velocidad de conexión.
		self.baudrate = baudrate
		# Longitud del byte.
		self.bytelength = bytelength
		# Paridad - Esta variable es un string que puede valer "Si" o "No".
		self.parity = parity
		# Bits de parada - Esta variable es un entero que puede valer 0 o
1.
		self.stopbits = int(stopbits)
		# Convertimos la variable port a una variable de tipo string.
		port = str(port)
		# La asignamos a la variable puerto.
		self.puerto = port

	# La siguiente función será la encargada de enviar la posición de acimut
	# y altura requerida por el usuario.
	# Tendremos que darle el acimut y altura que deseemos.
	def enviar_posicion(self, acimut, altura):
		# Para depurar los errores haremos uso de una sentencia del tipo
		# try-except. Así, si hay algún problema, el sistema saltará a
		# except en vez de detenerse.
		try:
			# Creamos la conexión enviar_serial. Los parámetros como el
			# puerto de la conexión, la velocidad de conexión y los bits
```

```
                # de parada serán los proporcionados por el constructor.
                enviar_serial = serial.Serial(self.puerto, self.baudrate,\
                timeout = self.stopbits)

                # Envíamos la posición utilizando los datos de acimut y
                # altura proporcionados al crear la función.
                enviar_serial.write("W%s %s" % (acimut, altura))

                # Enviamos un retorno de carro (salto de línea).
                enviar_serial.write("\r")

                # Cerramos la conexión.
                enviar_serial.close()

        # Si hay algún problema al crear la conexión el programa saltará
        aquí.
        except:
                # Mostraremos el mensaje 'Error en el envio' por pantalla.
                print "Error en el envio"

# La función parar_sistema detendrá el sistema si el usuario así lo desea.
def parar_sistema(self):
        # De nuevo haremos uso de una sentencia try-except para evitar que
        # el sistema se bloquee.
        try:
                # Crearemos la conexión parar_serial asignándole los valores
                # de configuración del constructor.
                parar_serial = serial.Serial(self.puerto, self.baudrate,\
                timeout = self.stopbits)

                # Enviamos el caracter 'S' al rotor.
                parar_serial.write("S")

                # Enviamos un retorno de carro (salto de línea).
                enviar_serial.write("\r")

                # Cerramos la conexión.
                parar_serial.close()

        # Si hay algún problema enviando el carácer el sistema seguirá por
        # aquí.
        except:
                # Mostraremos el mensaje 'Sistema no detenido' por pantalla
                # para avisar al usuario.
                print "Sistema no detenido"

# Funcion para comprobar la conexion del sistema.
# Comprobaremos el correcto funcionamiento de esta conexión cuando
# conectemos el dispositivo al programa.
# Esta función nos devolverá los valores de posición actuales del rotor.
def comprobar_conexion(self):
        # Encapsulamos las sentencias en un try-except. Así, si hay un
        # error, el sistema nos avisará de él en vez de detenerse.
```

```
try:
        # Creamos un objeto llamado comprobar_conexion del tipo
        # Serial.
        # Iniciamos la conexión asignándole la dirección del puerto
        # y la velocidad de conexión.
        comprobar_conexion = serial.Serial(self.puerto,\
        self.baudrate)
        # Cancelamos los bits de parada.
        comprobar_conexion.timeout = None

        # Envíamos el caracter C para que el sistema nos devuelva el
        # acimut.
        comprobar_conexion.write("C")

        # Enviamos una señal para que nos devuelva datos (line
        # feed).
        enviar_serial.write("\n")

        # En este punto el controlador debería devolvernos una
        # cadena de caracteres del tipo +0aaa".

        # Mostramos un aviso por pantalla. Nos implementamos la
        # recolección de los datos de vuelta porque no disponemos
        # del rotor.
        print "Nos devuelve el acimut actual"

        # Envíamos el caracter B para que el sistema nos devuelva la
        # altura.
        comprobar_conexion.write("B")

        # En este punto el controlador debería devolvernos un
        # cadena de caracteres del tipo +0aaa".
        # Volveremos a mostrar un aviso por pantalla llegados a este
        # punto.
        print "Nos devuelve la elevacion actual"

        # Enviamos una señal para que nos devuelva datos (line
        # feed).
        enviar_serial.write("\n")

        # Cerramos la conexión comprobar_conexion.
        comprobar_conexion.close()

# En el caso de que encontremos algún problema mostraremos un aviso
# por línea de comandos.
except:
        print "Error de conexion"
```

Bibliografía

tutorialspoint. (n.d.). *Python Tkinter place() Method*. Retrieved 11 de Noviembre de 2013 from Tutorials for Javamail API, MongoDB, Git, Swing, Objective C, Android, jMeter, Data Communication, MIS, AWT, SVN, TestNG, VBScript, MATLAB, EJB, IPv6, IPv4, E-Commerce, PostgreSQL, SQLite, SDLC, Assembly, Operating System, JasperReports, JSON, iOS, Design Pattern, VB.Net, Computer Fundamentals, JSF, C Sharp, Flex, GWT, PL_SQL, Eclipse, JUnit, Pascal, Maven, Scala, Spring, Struts 2, HTML5, ANT, iBATIS, log4j, Hibernate, JSP, JAVA, JDBC, AJAX, WAP, SQL, MySQL, C_C++, PERL, PHP, Ruby, Ruby on Rails, Python, HTML, XHTML, CSS, CGI, Shell, Unix, JavaScript, jQuery, Radius, UML, GPRS, GSM,i-Mode, WiFi, WML, WiMax, SOAP, UDDI, Socket, WSDL, Makefile, Prototype, Six Sigma, CMMI, EVM, PMP Exams.htm: http://www.tutorialspoint.com/python/tk_place.htm

Autonomous Nonprofit Organization "TV-Novosti". (15 de Agosto de 2013). *Air Force shuts down 'Space Fence' surveillance system — RT USA*. Retrieved 2 de Diciembre de 2013 from http://rt.com/: http://rt.com/usa/space-fence-shut-down-455/

Felix R. Hoots, R. L. (31 de Diciembre de 1988). *SPACETRACK REPORT NO. 3 - spacetrk.pdf.* (T. Kelso, Ed.) Retrieved 2 de Diciembre de 2013 from Celestrak: http://celestrak.com/NORAD/documentation/spacetrk.pdf

Finlay, J. (7 de Octubre de 2012). *PyGTK 2.0 Tutorial - pygtk2-tut.pdf.* Retrieved 10 de Noviembre de 2013 from PyGTK: http://www.pygtk.org/dist/pygtk2-tut.pdf
Free Software Fundation. (28 de Febrero de 2013). *GNU Lesser General Public License v3.0 - GNU Project - Free Software Foundation (FSF)*. Retrieved 11 de Noviembre de 2013 from El sistema operativo GNU: http://www.gnu.org/licenses/lgpl.html

Grayson, J. E. (2000). *Python and Tkinter Programming.* Greenwich: Manning Publications Co.

Kelso, D. T. (26 de Agosto de 2006). *CelesTrak: "FAQs: Two-Line Element Set Format"*. Retrieved 2 de Diciembre de 2013 from CelesTrak: http://celestrak.com/columns/v04n03/

Kelso, D. T. (2 de Diciembre de 2013). *CelesTrak: Current NORAD Two-Line Element Sets*. Retrieved 2 de Diciembre de 2013 from Celestrak: http://celestrak.com/NORAD/elements/

Kelso, D. T. (25 de Septiembre de 2013). *Dr. T.S. Kelso, CelesTrak WWW*. Retrieved 2 de Diciembre de 2013 from SpaceTrack: http://celestrak.com/webmaster.asp

Lutz, M. (2011). *Programming Python.* Sebastopol , Canada: O'Reilly Media, Inc.
PyPA. (10 de Noviembre de 2013). *Installation -- pip ...* Retrieved 27 de Noviembre de 2013 from pip -- pip 1.4.1 doc ...: http://www.pip-installer.org/en/latest/installing.html

Python Community. (1 de Enero de 2013). *Tkinter - Python Wiki.* Retrieved 8 de Noviembre de 2013 from Python Programming Language - Official Website: https://wiki.python.org/moin/TkInter

Python Software Foundation. (11 de Noviembre de 2013). *24.1. Tkinter -- Python interface to Tcl/Tk -- Python v2.7.6 documentation*. Retrieved 11 de Noviembre de 2013 from Python Programming Language - Official Website: http://docs.python.org/2/library/tkinter.html#packer-options

Python Software Foundation. (2013). *Download Python*. Retrieved 27 de Noviembre de 2013 from Python Programming Language - Official Website: http://www.python.org/getit/

Shipman, J. W. (10 de Octubre de 2009). ScrolledList: A Tkinter scrollable list widget. (N. M. Tech, Ed.) Socorro, Nuevo Mexico, Estados Unidos.

Shipman, J. W. (24 de Junio de 2013). Tkinter 8.5 reference: a GUI for Python. (N. M. Tech, Ed.) Socorro, Nuevo México, Estados Unidos.

Smith, Y. (10 de Julio de 2012). *NASA - July 12, 1962: The Day Information Went Global*. Retrieved 2 de Diciembre de 2013 from NASA: http://www.nasa.gov/topics/technology/features/telstar.html

SSN. (Diciembre de 2012). *USSTRATCOM Space Control and Space Surveillance (JSPoC) - U.S. Strategic Command*. Retrieved 2 de Diciembre de 2013 from Home - U.S. Strategic Command: http://www.stratcom.mil/factsheets/USSTRATCOM_Space_Control_and_Space_Surveillance/

The Raspberry Pi Foundation. (n.d.). *Downloads | Raspberry Pi*. Retrieved 1 de Diciembre de 2013 from Raspberry Pi | An ARM GNU/Linux box for $25. Take a byte!: http://www.raspberrypi.org/downloads

Wouters, T. (25 de Abril de 2010). *class - Python 'self' explained - Stack Overflow.* Retrieved 8 de Noviembre de 2013 from Stack Overflow: http://stackoverflow.com/questions/2709821/python-self-explained